# Computer Graphics (UCS505)

# Project on

# Obstacle Driving Game

**Submitted By**

Yuvraj Puri 102153041

**B.E. Third Year – COE**

**Submitted To:**

**MS. Jasmine Kaur**



**Computer Science and Engineering Department Thapar Institute of Engineering and Technology Patiala – 147001**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Description** | **Page No.** |
| 1. | Introduction to Project | 3 |
| 2. | Computer Graphics concepts used | 3 |
| 3. | User Defined Functions | 4 |
| 4. | Code | 7 |
| 5. | Output/ Screen shots | 12 |

**INTRODUCTION TO PROJECT**

Our project is a captivating Obstacle Driving Game, harnessing the capabilities of SDL2, stb\_image, and GLUT libraries, players traverse a dynamic landscape, skillfully maneuvering to evade obstacles.

**COMPUTER GRAPHICS CONCEPTS USED**

1. **OpenGL:** The foundation of our computer graphics project lies in OpenGL, a powerful graphics library that facilitates rendering 2D and 3D graphics in real-time. By harnessing OpenGL's rendering pipeline and extensive functionality, our project achieves high-performance graphics rendering across various platforms. OpenGL provides a versatile framework for implementing rendering techniques, enabling the creation of visually stunning environments and effects within our obstacle driving game.
2. **Transformation:** In computer graphics, transformation plays a pivotal role in manipulating objects within a scene. Our project utilizes transformation matrices to translate, rotate, and scale graphical elements, such as the vehicle and obstacles, within the game world. By applying transformations to geometric primitives, we can dynamically position and orient objects in response to user input, enhancing the interactive experience of our obstacle driving game.
3. **Rendering:** Rendering encompasses the process of generating visual output from geometric data within a computer graphics application. Leveraging OpenGL's rendering pipeline, our project efficiently converts geometric primitives into pixel-based images on the screen. Through techniques such as rasterization, we achieve realistic rendering of the game environment, including terrain, obstacles, and the player's vehicle. Rendering optimizations ensure smooth frame rates and immersive visuals, enhancing the overall gameplay experience.
4. **Data Structures:** Effective management of data structures is essential for organizing and manipulating graphical elements within a computer graphics project. Our obstacle driving game employs data structures such as vector, stacks, queues etc. to store information about obstacles, markings, bullets, image data. By leveraging efficient data structures, we optimize memory usage and reducing time complexity.
5. **Image Rendering:** Image rendering involves the process of displaying 2D images within a computer graphics application. Utilizing OpenGL and image loading libraries such as stb\_image, our project seamlessly integrates image rendering to enhance visual elements within the game. From textures applied to terrain surfaces to graphical user interface elements, image rendering reduces the need the draw complex structures.

**USER DEFINED FUCNTIONS**

1. void initGrid(int x, int y)

* Initializes coordinate points for game components like trees, markings, obstacles, and cars.
* Loads associated images during initialization to optimize performance.
* Ensures smooth rendering and efficient gameplay by establishing initial layout and preloading images.

1. void initMarking()

* The initMarking function are used to append the marking objects in their respective arrays i.e. left, right, and middle.
* void initObstacle()
* Populates the Obstacle stack with obstacle objects during initialization.
* Facilitates efficient management and distribution of obstacles across lanes.

1. void drawTree(Tree obj)

* Draws a tree object onto the canvas based on its coordinates.
* Utilizes mathematical functions to render a rectangle and a triangle, resembling a tree.

1. void drawGrass()

* Renders green-colored rectangles on the sides of the canvas to simulate a grassy area.
* Enhances environmental realism by adding visual elements that evoke the appearance of grass.

1. void drawRoad()

* Renders a black rectangular area on the canvas.
* Draws white lines to divide the area into three lanes.

1. void drawMarking()

* Renders markings from their arrays onto their respective lanes.
* Draws white-colored rectangles to represent the markings.

1. void drawCar()

* Utilizes car object coordinates to draw the car at its position.
* Utilizes loaded images to represent the car.
* Renders the car by projecting the image onto a rectangle, enhancing visual realism within the game environment.

1. void drawObstacle()

* Draws obstacle objects from the stack based on their coordinates.
* Utilizes a rectangle as a base for projecting obstacle images.
* Monitors collisions between the car and obstacles, ensuring accurate gameplay dynamics and obstacle avoidance.

1. void initBullet()

* Initializes bullet objects and loads them into an array.
* Prepares bullets for use within the game environment.

1. void drawBullet()

* Renders bullets based on their object coordinates using a sliced image.

 Utilizes loaded images to accurately represent bullets within the game environment.

1. void collisionDetection()

* Detects collisions between bullets and obstacles using the AAS collision detection algorithm.
* Removes the respective bullet and obstacle upon collision detection.
* Ensures accurate gameplay dynamics by eliminating collided objects, enhancing player immersion and challenge.

1. void showScore()

* Displays the in-game score using OpenGL's text rendering feature.
* Enhances player feedback and engagement by presenting real-time score updates.
* Utilizes text rendering capabilities to ensure clear and readable presentation of the score within the game environment.

1. void finalScore()

* Showcases the final score at the end screen using depth-rendering capabilities.
* Provides players with a summary of their performance upon completing the game.
* Utilizes advanced rendering features to ensure the final score is prominently displayed and visually appealing.

**CODE (Source.cpp)**

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

**#include "game.h"**

**#include "tree.h"**

**#include "marking.h"**

**#include "menu.h"**

**#include<iostream>**

**#define COLUMNS 120**

**#define ROWS 70**

**extern short FPS;**

**extern short sDirection;**

**extern short gamePaused;**

**extern short gameMain;**

**extern short crashed;**

**extern int score;**

**extern short finished;**

**extern int bullet;**

**extern int shoot;**

**extern int dc;**

**void timer\_callback(int);**

**void displayCallback();**

**void reshape\_callback(int, int);**

**void keyboard\_callback(int, int, int);**

**void keyboard\_callback2(unsigned char, int, int);**

**void init() {**

**glClearColor(0.0, 0.0, 0.0, 1.0);**

**initGrid(COLUMNS, ROWS);**

**score = 0;**

**}**

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

**glutInit(&argc, argv);**

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

**glutInitWindowSize(1536, 864);**

**glutCreateWindow("Obstacle Driving");**

**glutDisplayFunc(displayCallback);**

**glutReshapeFunc(reshape\_callback);**

**glutTimerFunc(0, timer\_callback, 0);**

**glutSpecialFunc(keyboard\_callback);**

**glutKeyboardFunc(keyboard\_callback2);**

**init();**

**glutMainLoop();**

**return 0;**

**}**

**int index = 0;**

**void displayCallback() {**

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

**if (gamePaused == 0 && gameMain == 0 && crashed == 0 && finished ==0) {**

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

**drawGrass();**

**drawRoad();**

**drawMarking();**

**drawCar();**

**drawObstacle();**

**if (shoot == 1) {**

**initBullet();**

**}**

**drawBullet();**

**if (dc == 1) {**

**drawExplosion();**

**}**

**collisionDetection();**

**showScore();**

**}**

**else if (gameMain == 1) {**

**glColor3f(1, 1, 1);**

**drawMain();**

**showScore();**

**}**

**else if (crashed == 1) {**

**glColor3f(1, 1, 1);**

**drawCrashed();**

**finalScore();**

**}**

**else if (finished == 1) {**

**glColor3f(1, 1, 1);**

**drawFinished();**

**finalScore2();**

**}**

**else {**

**glColor3f(1,1,1);**

**drawPause();**

**}**

**glutSwapBuffers();**

**}**

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

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

**glMatrixMode(GL\_PROJECTION);**

**glLoadIdentity();**

**glOrtho(0.0, COLUMNS, 0.0, ROWS, -1.0, 1.0);**

**glMatrixMode(GL\_MODELVIEW);**

**}**

**void timer\_callback(int) {**

**glutPostRedisplay();**

**glutTimerFunc(1000/FPS,timer\_callback,0);**

**}**

**void keyboard\_callback(int key, int, int) {**

**std::cout << key << std::endl;**

**switch (key)**

**{**

**case GLUT\_KEY\_LEFT:**

**if (sDirection != LEFTSTOP)**

**sDirection = LEFT;**

**break;**

**case GLUT\_KEY\_RIGHT:**

**if (gameMain == 1) {**

**gameMain = 0;**

**}**

**if (sDirection != RIGHTSTOP)**

**sDirection = RIGHT;**

**break;**

**}**

**}**

**void keyboard\_callback2(unsigned char key, int, int) {**

**std::cout << key << std::endl;**

**switch (key)**

**{**

**case 13:**

**if (gameMain == 1) {**

**gameMain = 0;**

**}**

**else if (gamePaused == 1) {**

**gamePaused = 0;**

**}**

**else if (crashed == 1) {**

**crashed = 0;**

**score = 0;**

**initGrid(COLUMNS, ROWS);**

**}**

**else if(finished == 1) {**

**finished = 0;**

**score = 0;**

**initGrid(COLUMNS, ROWS);**

**}**

**break;**

**case 27:**

**if (gameMain == 1) {**

**exit(0);**

**break;**

**}**

**else if (gameMain == 0 && gamePaused == 0) {**

**gamePaused = 1;**

**}**

**break;**

**case 'q':**

**if (gamePaused == 1) {**

**gamePaused = 0;**

**gameMain = 1;**

**initGrid(COLUMNS,ROWS);**

**}**

**else if (crashed == 1) {**

**crashed = 0;**

**gameMain = 1;**

**initGrid(COLUMNS, ROWS);**

**score = 0;**

**}**

**else if(finished == 1) {**

**gameMain = 1;**

**finished = 0;**

**score = 0;**

**initGrid(COLUMNS, ROWS);**

**}**

**break;**

**case 32:**

**if (bullet >0) {**

**bullet--;**

**shoot = 1;**

**}**

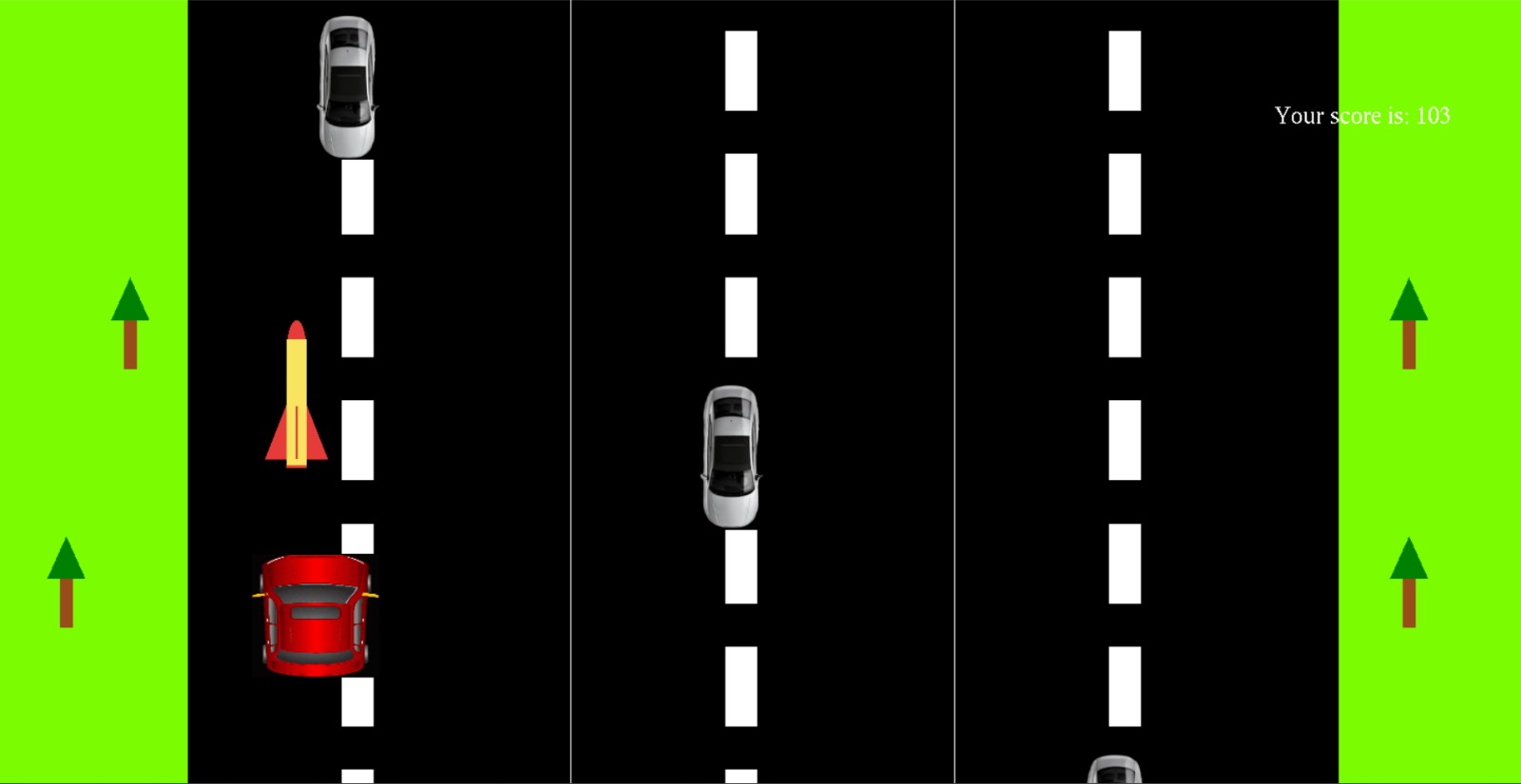
**}**

**}**

**SCREENSHOTS**

1.

2.

****

3.

4.