Computer Graphics (UCS505)

Project Name-F1 Car Racing

Branch

B.E. 3rd Year – COE/CSE

Submitted By –

Suhani Mathur (10220522)

Shivansh Tuteja (102203513)

Shivansh Gupta (102203508)

Submitted To –
Ms. Jasmine Kaur

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

INTRODUCTION

Project Overview:

This project simulates a **Formula** 1 racing experience in a 3D environment. Built with **OpenGL and C/C++**, the game renders detailed race tracks and vehicles, allowing players to choose between two distinct tracks. The game includes a **Heads-Up Display (HUD)** that keeps players informed of their performance, fostering competitive and replayable gameplay.

Players can control their car in real-time, monitor lap stats, and reset the race in case of crashes. The design focuses on game physics, rendering optimization, and responsive controls to create a smooth racing experience.

Scope of the Project:

- Create a 3D F1 racing game using OpenGL.
- Design and render two unique race tracks.
- Implement smooth car movement and directional control.
- Add crash reset functionality using the `R` key.
- Show current lap time, best lap, and lap number via HUD.
- Enable menu navigation or exit using the `Esc` key.
- Manage game states like racing, paused, and reset.
- Ensure real-time rendering with a consistent game loop.

USER-DEFINED FUNCTIONS

Category	Function	Description
Main (main.c)	main(int argc, char** argv)	Entry point. Initializes GLUT/GLEW, sets callbacks, enters main loop.
	display()	Renders menu or racing scene. Sets camera and overlays HUD.
	reshape(int width, int height)	Updates viewport and perspective when window is resized.
	keyboardDown(unsigned char key, int x, int y)	Handles key presses, routes based on state.
	keyboardUp(unsigned char key, int x, int y)	Handles key releases for car controls.
	specialKeyDown(int key, int x, int y)	Handles special key presses (e.g., arrow keys).
	cleanup()	Cleans up on exit.
Game Logic (game.c)	initGame()	Initializes car and lap state for selected track.
	startGame(TrackType type)	Starts the game with selected track.
	setupCamera()	Configures third-person camera behind the car.
	updateGame(int value)	Game loop callback. Updates car, timers, and lap detection.
	switchTrack(TrackType newType)	Switches between different track types.
	renderMenu(int w, int h)	Draws the main menu with options and instructions.
	renderHUD(int w, int h)	Draws lap time and other race metrics as HUD.
	handleMenuKeyPress(unsigned char key)	Handles menu input (Enter to start, Esc to quit).
	handleMenuSpecialKey(int key)	Allows track switching with arrow keys in menu.
	handleRacingKeyPress(unsigned char key)	Handles in-race controls like WASD, reset, or exit.
	handleRacingSpecialKey(int key)	Placeholder for special keys during racing.
Car Physics (car.c)	initCar(Car* car)	Sets car position, speed, orientation, and control flags.
	calculateCarCorners(...)	Computes corner coordinates for collision

		detection.
	updateCar(Car* car, float deltaTime)	Applies physics: movement, turning, and friction. Checks track collision.
	isPositionOnTrack(float x, float z)	Verifies if car is within allowed track bounds.
	renderCar(const Car* car)	Draws the car at its current location and rotation.
	setCarControls(Car* car, int key, int state)	Updates movement flags based on input.
Track Rendering	renderRectTrack() / renderRoundTrack()	Renders full track surface and decorations.
	renderRectGuardrails() / renderRoundGuardrails()	Renders guardrails on the sides of the track.
	isPositionOnRectTrack(float x, float z)	Validates car position on rectangular track.
	isPositionOnRoundTrack(float x, float z)	Validates car position on circular/rounded track.
	drawWallRect(...) / drawWallRound(...)	Helper to draw straight or curved wall sections.
	renderCornerLineSegmentRound(...)	Draws curved lines for round tracks (edges, markings).
	renderCornerSurfaceSegmentRound(...)	Draws curved track surfaces (e.g., corners).

CODE SNIPPETS

main.c

```
#include <stdio.h>
#include <GL/glew.h>
#include <GL/freeglut.h>
#include "game.h"
#include "track_rect.h"
#include "track_round.h"

void display();
void reshape(int width, int height);
void keyboardDown(unsigned char key, int x, int y);
void keyboardUp(unsigned char key, int x, int y);
void specialKeyDown(int key, int x, int y);
void cleanup();

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(1280, 720);
    glutCreateWindow("F1 Racer");

    GLenum err = glewInit();
    if (GLEW_OK != err) {
        fprintf(stderr, "Error initializing GLEW: %s\n",
glewGetErrorString(err));
        return 1;
    }

    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glClearColor(0.1f, 0.3f, 0.7f, 1.0f);

    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboardDown);
    glutKeyboardUpFunc(keyboardUp);
    glutSpecialFunc(specialKeyDown);
    glutCloseFunc(cleanup);

    glutTimerFunc(FRAME_TIME_MS, updateGame, 0);

    printf("\n--- CONTROLS ---\n");
    printf(" Menu:\n");
    printf("   UP/DOWN Arrows: Select Track\n");
    printf("   ENTER: Start Race\n");
    printf(" Racing:\n");
    printf("   W/S: Accelerate/Brake\n");
    printf("   A/D: Turn Left/Right\n");
    printf("   R: Reset Race\n");
    printf(" General:\n");
    printf("   ESC: Return to Menu / Exit\n");
    printf("-----------------\n\n");

    glutMainLoop();

    return 0;
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    if (currentGameState == STATE_MENU) {
        renderMenu(glutGet(GLUT_WINDOW_WIDTH),
glutGet(GLUT_WINDOW_HEIGHT));
    } else {
        glMatrixMode(GL_PROJECTION); glLoadIdentity();
        gluPerspective(50.0f, (float)glutGet(GLUT_WINDOW_WIDTH)
/ (float)glutGet(GLUT_WINDOW_HEIGHT), 0.1f, 600.0f);
        glMatrixMode(GL_MODELVIEW); glLoadIdentity();
        setupCamera();

        if (selectedTrackType == TRACK_RECT) {
            renderRectTrack();
            renderRectGuardrails();
        } else {
            renderRoundTrack();
            renderRoundGuardrails();
        }
        renderCar(&playerCar);

        renderHUD(glutGet(GLUT_WINDOW_WIDTH),
glutGet(GLUT_WINDOW_HEIGHT));
    }

    glutSwapBuffers();
}

void reshape(int width, int height) {
    if (height == 0) height = 1;
    float aspect = (float)width / (float)height;
    glViewport(0, 0, width, height);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(50.0f, aspect, 0.1f, 600.0f);
    glMatrixMode(GL_MODELVIEW);
}

void keyboardDown(unsigned char key, int x, int y) {
    (void)x; (void)y;

    if (currentGameState == STATE_MENU) {
        handleMenuKeyPress(key);
    } else {
        handleRacingKeyPress(key);
    }
}

void keyboardUp(unsigned char key, int x, int y) {
    (void)x; (void)y;

    if (currentGameState == STATE_RACING) {
        if (key == 'w' || key == 'W' || key == 'a' || key ==
'A' || key == 's' || key == 'S' || key == 'd' || key == 'D') {
            setCarControls(&playerCar, key, 0);
        }
    }
}

void specialKeyDown(int key, int x, int y) {
    (void)x; (void)y;
    if (currentGameState == STATE_MENU) {
        handleMenuSpecialKey(key);
    } else {
        handleRacingSpecialKey(key);
    }
}

void cleanup() {
    printf("Exiting application...\n");
}
```

game.c

```
#include "game.h"
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <limits.h>

#include "track_rect.h"
#include "track_round.h"

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

#define DEG_TO_RAD(angle) ((angle) * M_PI / 180.0f)

GameState currentGameState = STATE_MENU;
TrackType selectedTrackType = TRACK_RECT;
int menuSelectionIndex = 0;
Car playerCar;
int lapStartTimeMs = 0;
int currentLapTimeMs = 0;
int lastLapTimeMs = 0;
int bestLapTimeMs = INT_MAX;
int crossedFinishLineMovingForwardState = 0;
```

```
void switchTrack(TrackType newType) {
    selectedTrackType = newType;
}

void initGame() {
    initCar(&playerCar);

    lapStartTimeMs = glutGet(GLUT_ELAPSED_TIME);
    currentLapTimeMs = 0;
    lastLapTimeMs = 0;
    bestLapTimeMs = INT_MAX;

    float finishLineXStart, finishLineXEnd;
    if (selectedTrackType == TRACK_RECT) {
        finishLineXStart = RECT_FINISH_LINE_X_START;
        finishLineXEnd = RECT_FINISH_LINE_X_END;
    } else {
        finishLineXStart = ROUND_FINISH_LINE_X_START;
        finishLineXEnd = ROUND_FINISH_LINE_X_END;
    }

    crossedFinishLineMovingForwardState = (playerCar.z >=
FINISH_LINE_Z &&
                                           playerCar.x >=
finishLineXStart &&
                                           playerCar.x <=
finishLineXEnd);

    printf("Game Initialized for Track Type %d. Start time:
%dms. Crossed Flag: %d\n",
           selectedTrackType, lapStartTimeMs,
crossedFinishLineMovingForwardState);
}

void startGame(TrackType type) {
    printf("Starting game with Track Type %d\n", type);
    selectedTrackType = type;
    initGame();
    currentGameState = STATE_RACING;
    glutPostRedisplay();
}

void setupCamera() {
    float followDistance = 10.0f;
    float followHeight = 5.0f;
    float lookAtHeightOffset = 0.5f;

    float carAngleRad = DEG_TO_RAD(playerCar.angle);
    float camX = playerCar.x - followDistance *
sinf(carAngleRad);
    float camY = playerCar.y + followHeight;
    float camZ = playerCar.z - followDistance *
cosf(carAngleRad);

    float lookAtX = playerCar.x;
    float lookAtY = playerCar.y + lookAtHeightOffset;
    float lookAtZ = playerCar.z;

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(camX, camY, camZ, lookAtX, lookAtY, lookAtZ,
0.0f, 1.0f, 0.0f);
}

void updateGame(int value) {
    if (currentGameState != STATE_RACING) {
        glutTimerFunc(FRAME_TIME_MS, updateGame, 0);
        glutPostRedisplay();
        return;
    }

    (void)value;

    int timeNowMs = glutGet(GLUT_ELAPSED_TIME);

    updateCar(&playerCar, FRAME_TIME_SEC);

    if (timeNowMs >= lapStartTimeMs) {
        currentLapTimeMs = timeNowMs - lapStartTimeMs;
    } else {
        lapStartTimeMs = timeNowMs;
        currentLapTimeMs = 0;
    }

    float carZ = playerCar.z;
    float carPrevZ = playerCar.prev_z;
    float carX = playerCar.x;
    int movingForward = (playerCar.speed > 0.1f);

    float finishLineXStart, finishLineXEnd;
    if (selectedTrackType == TRACK_RECT) {
        finishLineXStart = RECT_FINISH_LINE_X_START;
        finishLineXEnd = RECT_FINISH_LINE_X_END;
    } else {
        finishLineXStart = ROUND_FINISH_LINE_X_START;
        finishLineXEnd = ROUND_FINISH_LINE_X_END;
    }
    int withinFinishLineX = (carX >= finishLineXStart && carX
<= finishLineXEnd);

    if (carPrevZ < FINISH_LINE_Z && carZ >= FINISH_LINE_Z &&
movingForward && withinFinishLineX) {
        if (crossedFinishLineMovingForwardState == 1) {
            lastLapTimeMs = currentLapTimeMs;
            if (lastLapTimeMs > 0 && lastLapTimeMs <
bestLapTimeMs) {
                bestLapTimeMs = lastLapTimeMs;
            }
            lapStartTimeMs = timeNowMs;
            currentLapTimeMs = 0;
        } else {
            crossedFinishLineMovingForwardState = 1;
            lapStartTimeMs = timeNowMs;
            currentLapTimeMs = 0;
        }
    } else if (carPrevZ >= FINISH_LINE_Z && carZ <
FINISH_LINE_Z && withinFinishLineX) {
        crossedFinishLineMovingForwardState = 0;
    }

    glutPostRedisplay();
    glutTimerFunc(FRAME_TIME_MS, updateGame, 0);
}
```

game.h

```
#ifndef GAME_H
#define GAME_H
```

```c
#include "car.h"

// --- Game States ---
typedef enum {
    STATE_Mz ENU,      // Menu state
    STATE_RACING     // Racing state
} GameState;

// --- Track Types ---
typedef enum {
    TRACK_RECT,      // Rectangular track
    TRACK_ROUNDED    // Rounded corners track
} TrackType;

// --- Menu Selection ---
#define NUM_TRACK_OPTIONS 2

// --- Frame Timing ---
#define FRAME_RATE 60
#define FRAME_TIME_MS (1000 / FRAME_RATE)
#define FRAME_TIME_SEC (1.0f / FRAME_RATE)

// --- Global Variables ---
extern GameState currentGameState;
extern TrackType selectedTrackType;
extern int menuSelectionIndex;
extern Car playerCar;

// Timer variables for lap timing
extern int lapStartTimeMs;
extern int currentLapTimeMs;
extern int lastLapTimeMs;
extern int bestLapTimeMs;
extern int crossedFinishLineMovingForwardState;

// --- Function Declarations ---
void initGame();
void updateGame(int value);
void setupCamera();
void startGame(TrackType type);
void switchTrack(TrackType newType);

void renderMenu(int windowWidth, int windowHeight);
void renderHUD(int windowWidth, int windowHeight);

void handleMenuKeyPress(unsigned char key);
void handleMenuSpecialKey(int key);
void handleRacingKeyPress(unsigned char key);
void handleRacingSpecialKey(int key);

#endif // GAME_H
```

how to compile

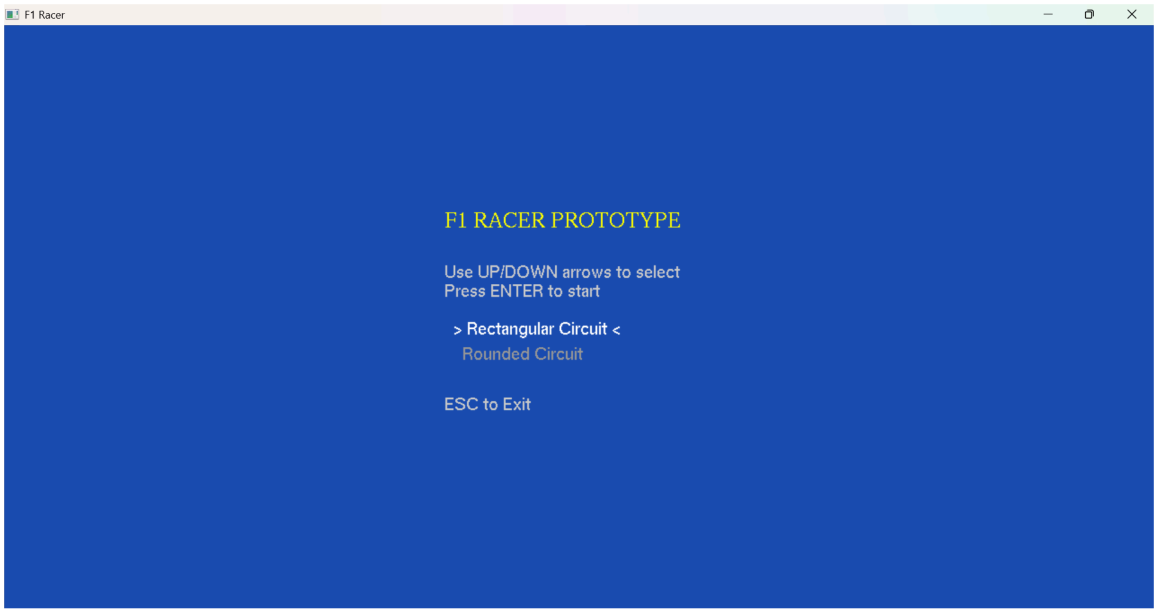
“compiling”

```
gcc -Wall -Wextra -O2 -c src/game.c -o obj/game.o
gcc -Wall -Wextra -O2 -c src/main.c -o obj/main.o
gcc -Wall -Wextra -O2 -c src/car.c -o obj/car.o
gcc -Wall -Wextra -O2 -c src/track_rect.c -o obj/track_rect.o
gcc -Wall -Wextra -O2 -c src/track_round.c -o obj/track_round.o
```

“linking”

```
gcc obj/main.o obj/game.o obj/car.o obj/track_rect.o
obj/track_round.o -o bin/game.exe -Llib -lfreeglut -lglew32
-lopengl32 -lm -lglu32 -mwindows
```

SCREENSHOTS


F1 Racer

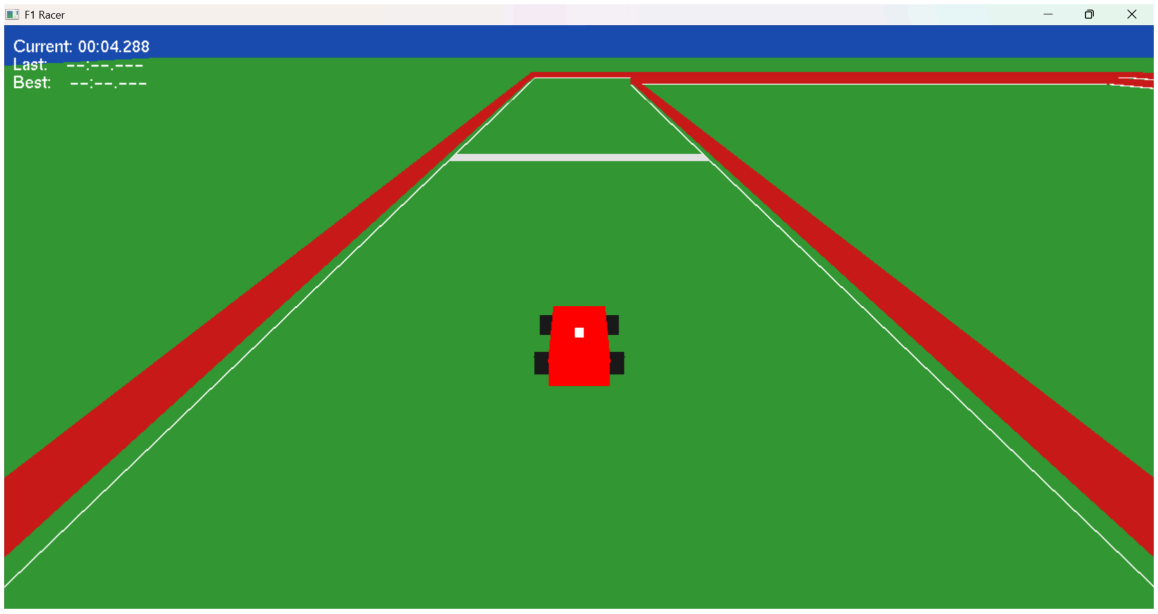
F1 RACER PROTOTYPE

Use UP/DOWN arrows to select
Press ENTER to start

> Rectangular Circuit <
Rounded Circuit

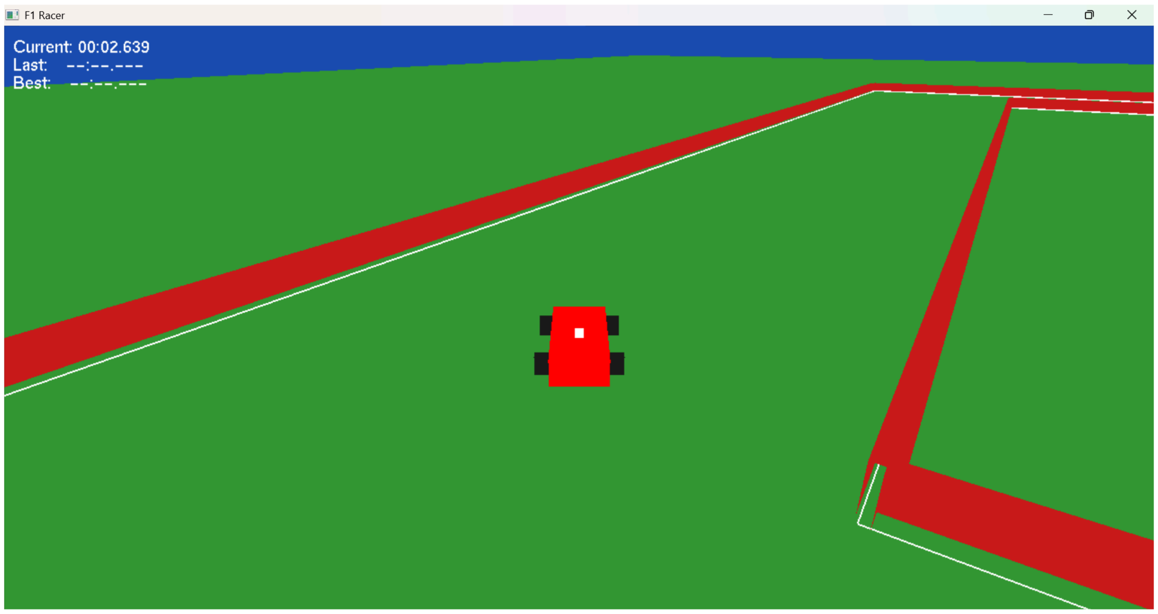
ESC to Exit

Opening page & menu of the F1 Racer

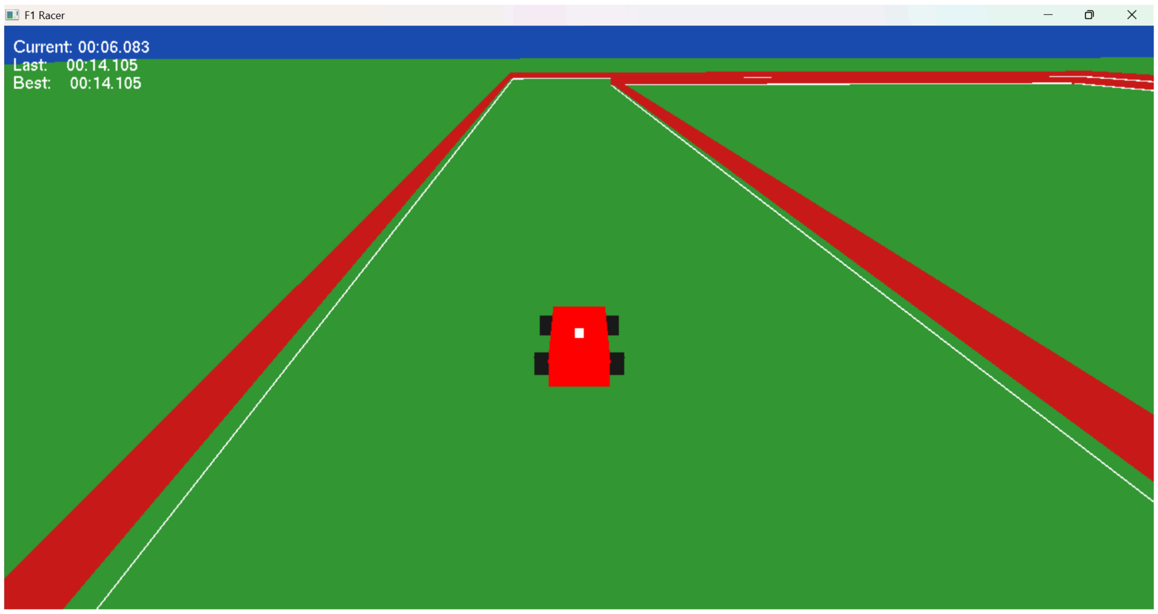

F1 Racer

```
Current: 00:04.288
Last:    --:--.---
Best:    --:--.---
```

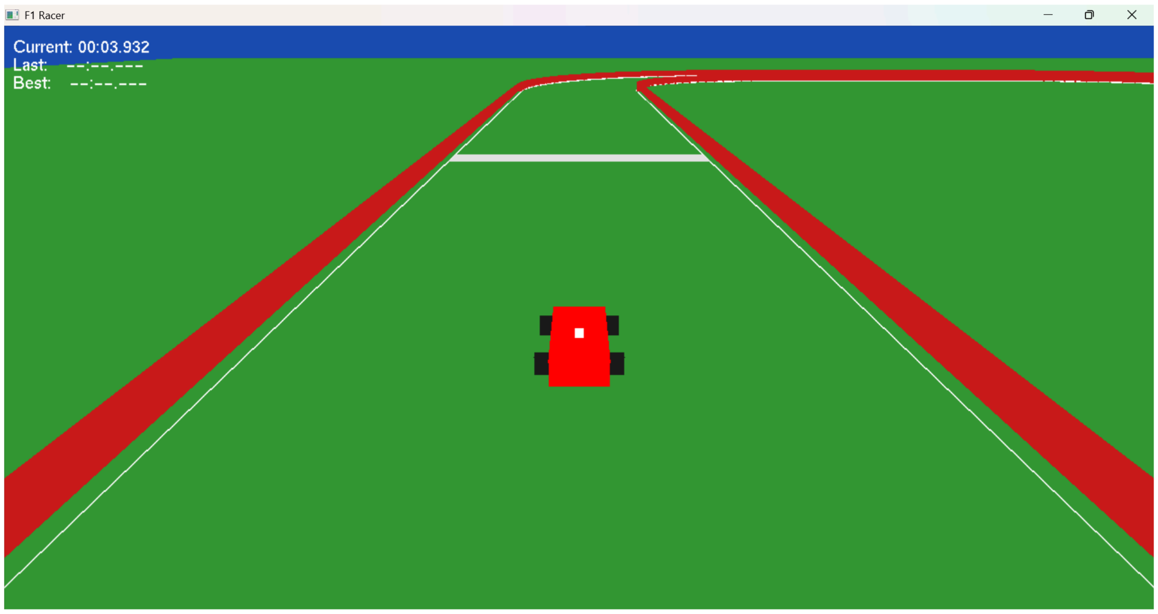
In-game view rectangular track starting position



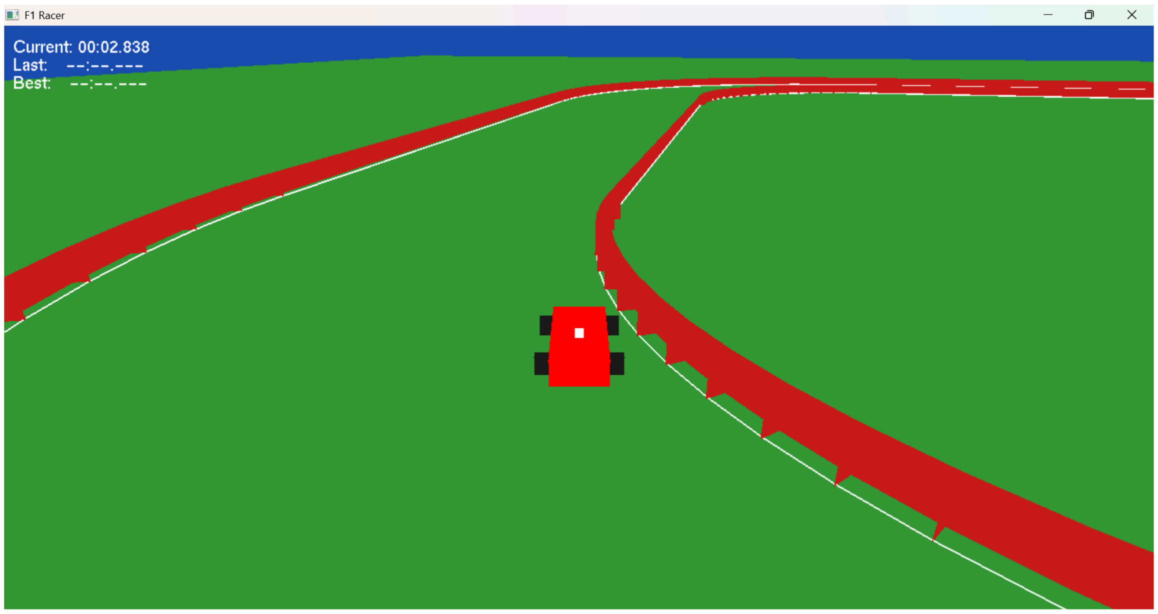
In-game view turn 1 rectangular track



In-game view rectangular track with HUD



In-game view round track starting position



In-game view round track turn 1

References

- **Github Repository**
  https://github.com/Shivansh12t/f1-racing-cg

- **OpenGL Documentation:**
  https://www.opengl.org/Documentation/Documentation.html

- **OpenGL Tutorials by Joey de Vries (LearnOpenGL)**
  https://learnopengl.com/In-Practice/2D-Game/Breakout

- **OpenGL Wiki (Official)**
  https://www.opengl.org/wiki/

- **LearnOpenGL - 3D Transformations**
  https://learnopengl.com/Getting-started/Transformations