Simulating Car Parking Experience

CS352: Computer Graphics & Visualization Lab

Project Report

Course Instructor:
Dr. Somnath Dey

Submitted By:

Boddupalli Karthik - 200001016
Karri Trived - 200001033
M. Sai Akshay Reddy - 200001049

Introduction

As the title suggests, our goal is to build a 3D model of a ‘Car Parking Lot’ where each car can be parked in an assigned slot. For this we need:

1. Layout of a parking lot which typically contains:
    a) Separate slots where each car can be parked
    b) Roads/Lanes
    c) Dividers dividing a set of slots
2. Multiple Cars

All of this should be rendered in 3D. Some of the features we want to implement are:

1. The whole model can be viewed from all angles and the user should be able to control it.
2. Each car should be distinguishable and should have individual control of its movement.
3. Stating the obvious but a car shouldn’t pass through another car/divider and should stop upon colliding.
4. Adjusting the brightness of the scene by having at least ‘Light’ and ‘Dark’ modes.

Above mentioned details should give a pretty brief idea of the simulation we are building and now moving on from ‘what to build’ to ‘how to build’, below are the technical specifications of the project:

Specifications

Technical specifications –

- Open Graphics Library (OpenGL)
- GNU C++ Library
- OpenGL Utility Toolkit Library (GLUT)

How to control the cars –

- To move a car first you must select the which car you are willing to move.
- Up key – forward movement of car
- Down key – backward movement of car
- Left Key – Car rotates in anti-clockwise direction
- Right Key – Car rotates in clockwise direction

How to control the camera –

- ‘A’ – Camera moves left
- ‘D’ – Camera moves right
- ‘S’ – Camera moves away
- ‘W’ – Camera moves near
- ‘T’ – Top view of the parking lot
- Up key – Moves in forward direction
- Down Key – Moves in backward direction
- Left Key – Turns left
- Right Key – Turns right

- Press ‘L’ to shift to dark mode and ‘l’ to switch back to light mode.
- After Parking the car, the user can press ‘Q’ to quit.

Functionalities Implemented

The main objective of our project is to simulate a parking lot in which user can park any car in any empty parking slot. The main functionalities and their implementation are as follows:

1) Collision with other Cars and Dividers

Description:

When a car collides with another car or divider, movement of the car in that direction becomes restricted and the car can move in any other direction until collision does not occur again.

Implementation:

- Collision of a car with a divider: As the divider is static object, it has a boundary and whenever our car tries to pass that boundary, we simply restrict the movement of the car in that direction. This cane be done by checking whether car coordinates are present inside the boundary enclosed by the divider.
- Collision of a car with a car: We used **“Separate axis Theorem”** to check the collision of cars. It simply checks if there exists a line between two polygons; if not then there is a potential intersection and so the movement of the car is restricted. It is implemented in **‘cars_intersecting’** function.

2) Different angles from which the user can view the parking lot

Description:

We can view the parking lot from different angles and the user can control it. He can also press ‘T’ for the top view.

Implementation:

It is implemented in **‘moveMeFlat’** and **‘orientMe’** functions. The former function uses W, A, S, D keys to move the camera in near, left, away, and right. The latter function uses arrow keys to move camera in forward, backward directions and rotate the camera in clockwise and anti-clockwise directions.

3) Movement of Cars

Description:

After selecting a particular car from the menu, the user can move that car using the ‘arrow keys’ as mentioned earlier in the specifications section.

Implementation:

This functionality was implemented in **‘movecar’** function. First it checks if there is any possible collision and takes the counter measure. Next for each of the movement keys, based on the present location and direction of the car we calculate dx and dz values for suitable translation and angle dθ for suitable rotation.

4) Lighting Effect

Description:

The user can change between light and dark modes by pressing ‘L’ and ‘l’ keys on the keyboard.

Implementation:

It is implemented in **‘initialize’** function.

5) Text Rendering

Description:

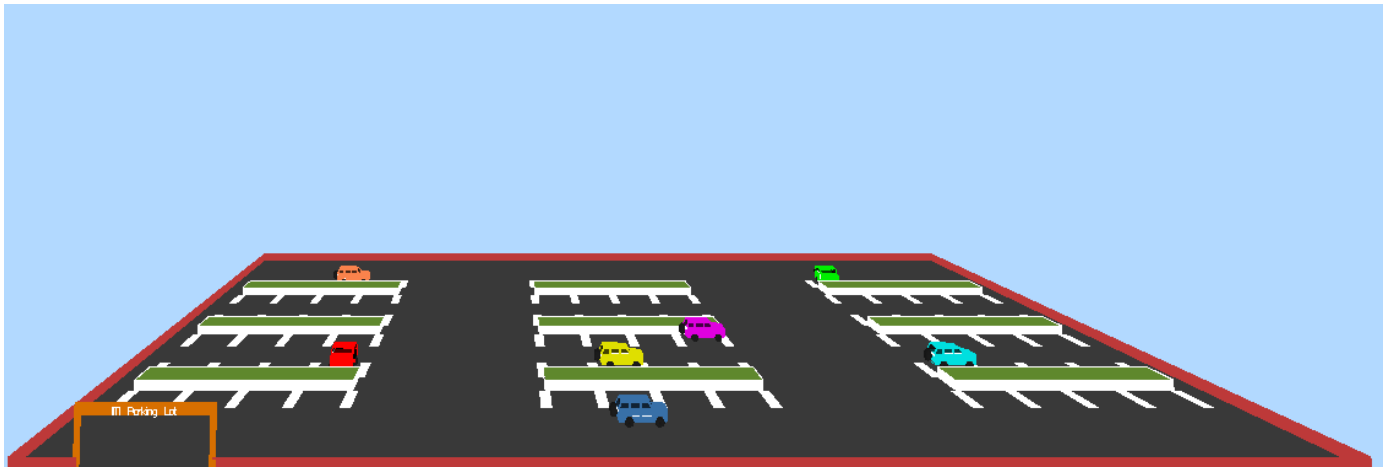
Each car has its brand name written on the rear side of the car.

Implementation:

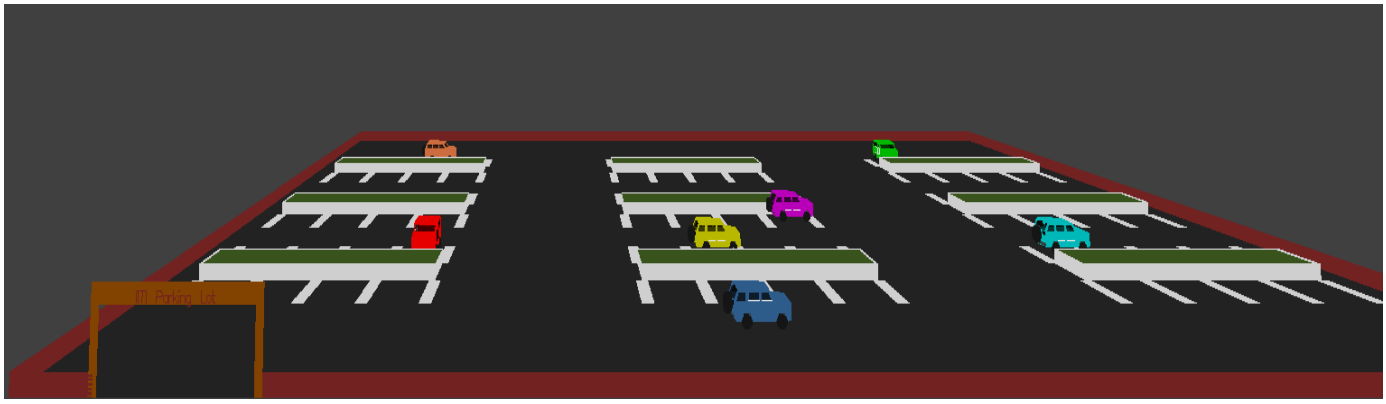
It is implemented in **‘drawstring’** function. It uses the inbuilt **‘glutStrokeCharacter’** function to print each character from the string.

Output

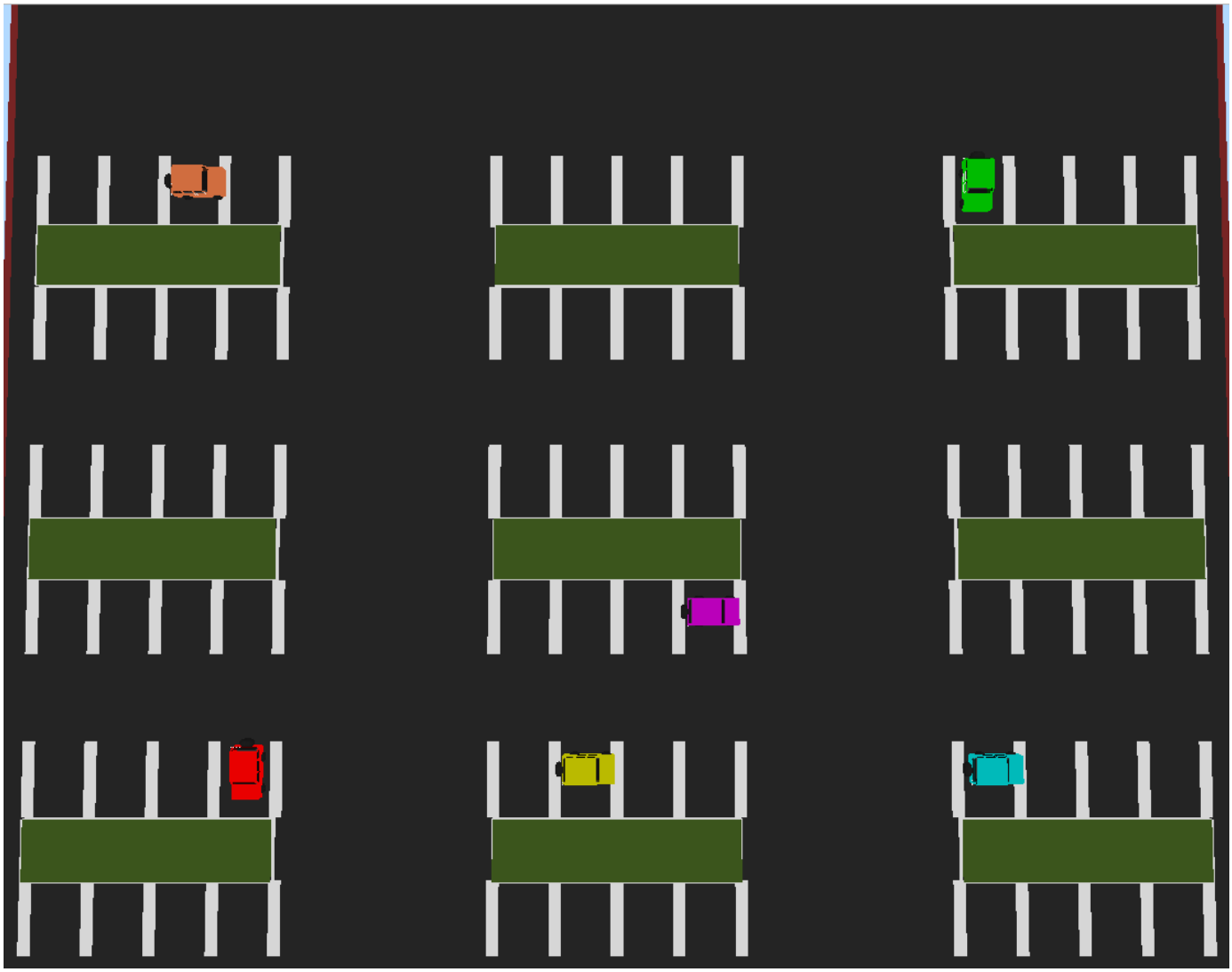
1) Overall view in Light Mode



2) Overall view in Dark Mode

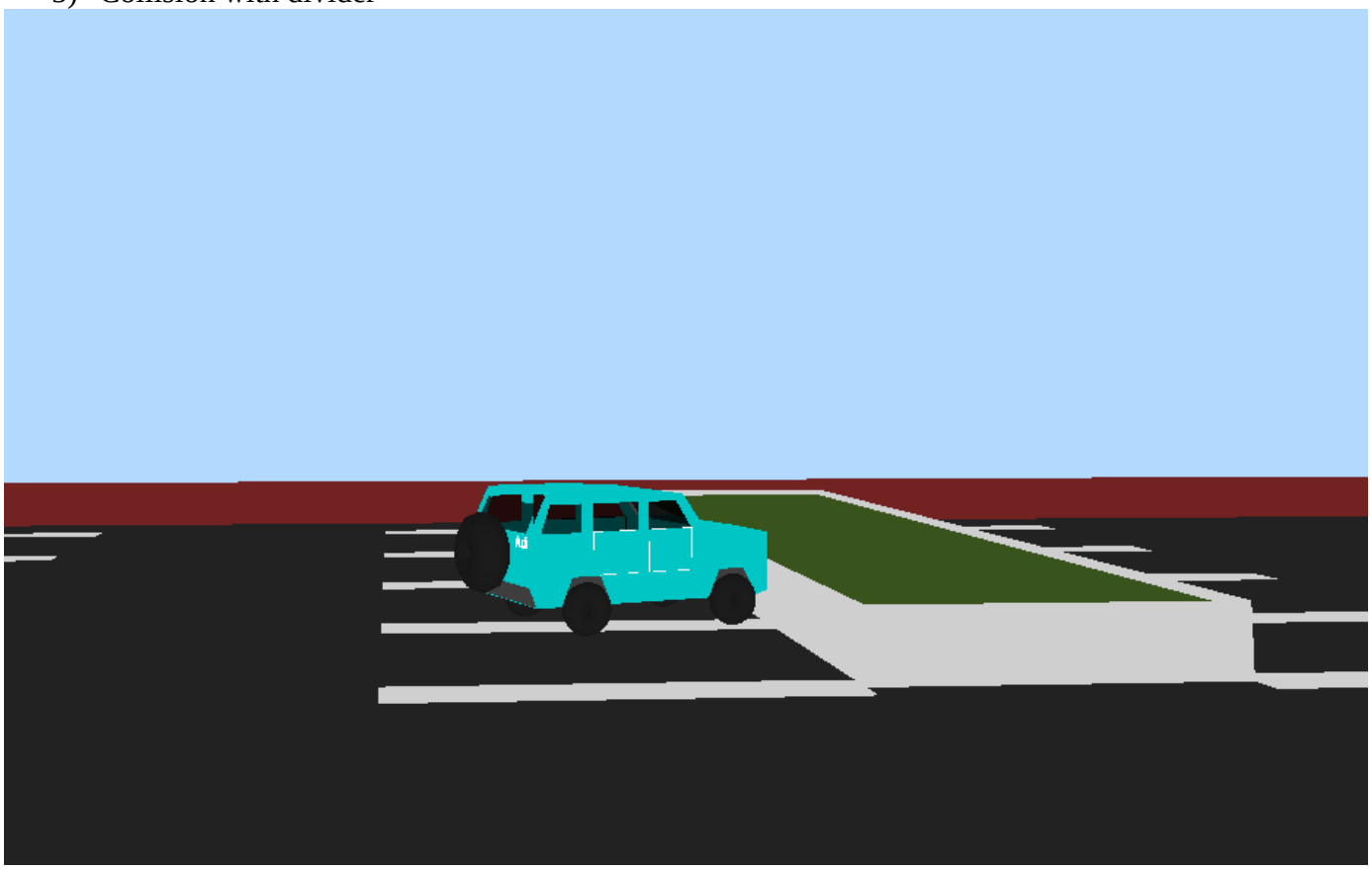


3) Top view

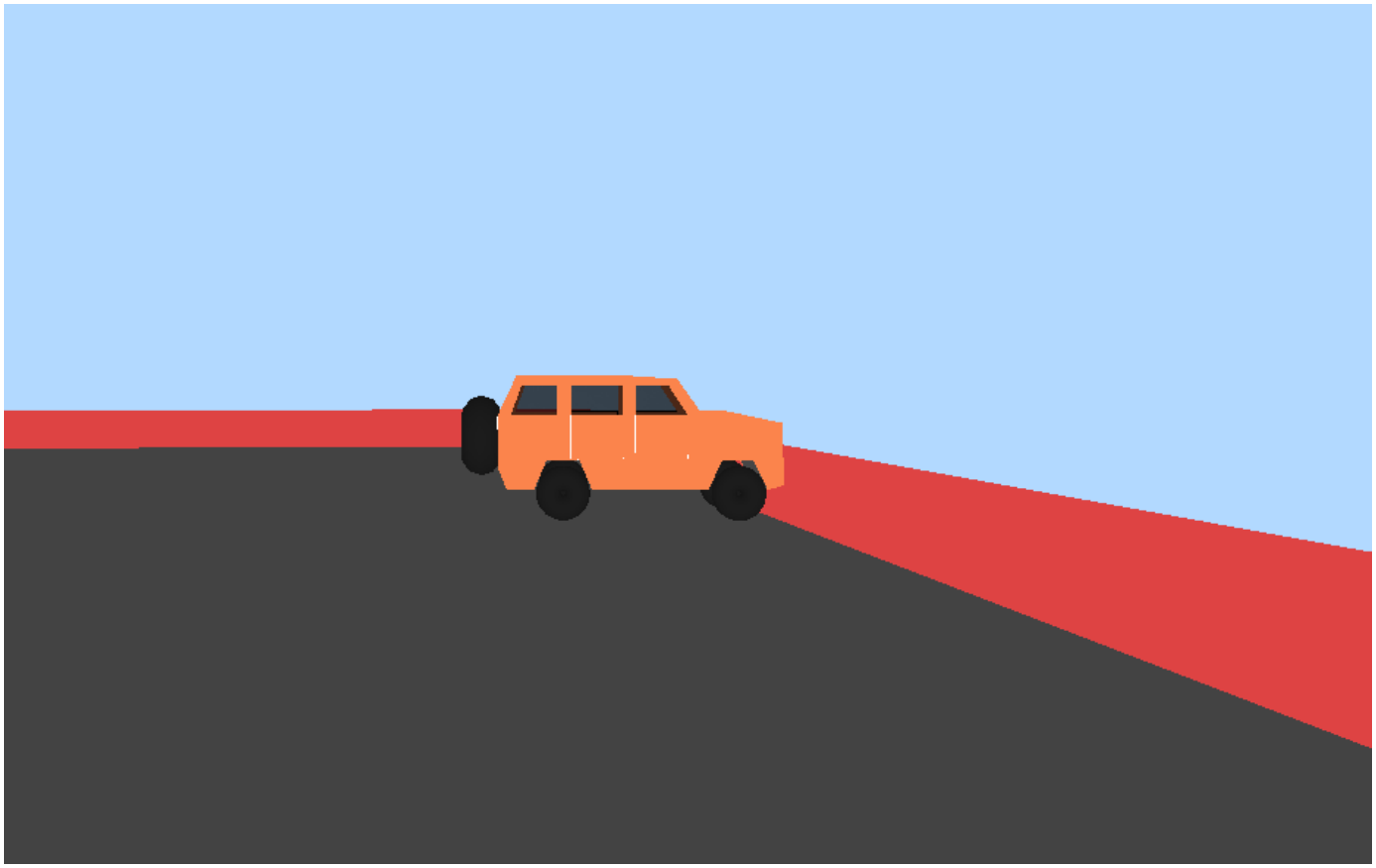


4) Backview of car

5) Collision with divider



6) Collision with fence



7) Collision with Car