ONTARIO TECH UNIVERSITY
FACULTY OF SCIENCE, COMPUTER SCIENCE

Project Group: 1 April 6th, 2024

Simulation and Modelling Course Project: Simulating Car Motion with Rigid Bodies and Centripetal Force using Python

by

Edison Lei(100750499) Justin Marsh (100792009)

Link to github: https://github.com/justin-marsh/car-simulation.

Our Simulation and Modeling project revolves around the development of a realistic car motion simulation, using Python and Pygame. By applying principles of rigid body dynamics and centripetal force, we aim to create an immersive experience that accurately depicts the behavior of a car under various driving conditions. This project not only serves as a demonstration of Python programming skills but also offers an opportunity to delve deep into the complexities of car motion within a virtual environment. Let's explore how we coded the simulation and the key features we incorporated.

1 Methodology

Our methodology involved the following steps:

Model Selection: We chose a suitable 3D model for the car and the racetrack, which serve as the primary entities in our simulation. These models provide the visual representation of the car's dynamics and the environment in which it operates.

Entity Creation: Using Ursina's Entity class, we instantiated objects representing the car and the racetrack, specifying parameters such as model, scale, position, and texture. This step established the visual components of our simulation.

Physics Implementation: We defined the physics parameters and state variables governing the behavior of the car. These included parameters such as maximum velocity, acceleration, deceleration, steering capabilities, and gravity. We translated these parameters into Python code, implementing update rules to simulate the car's movement and rotation based on user input and environmental conditions.

Integration of Differential Equations: Each state variable of the car (e.g., velocity, position, rotation) was associated with a set of differential equations representing its dynamic evolution over time. These equations encapsulate the fundamental principles of motion, including Newton's laws and rotational dynamics. By integrating these equations into our code, we ensured that the car's behavior closely adhered to the laws of physics.

User Interaction: We implemented functionality to enable user interaction with the simulation. This included capturing user input for acceleration, braking, and steering, as well as updating the camera position to provide a dynamic viewpoint of the car's motion.

2 Update Rules

Car Movement: The car's velocity is updated based on user input for acceleration, braking, and steering. Acceleration and deceleration are applied along the car's forward direction, while steering inputs modify the car's lateral velocity component.

Car Rotation: Steering inputs result in changes to the car's rotation, simulating the turning motion. The rotation speed parameter governs the rate at which the car rotates in response to steering commands. The turning mechanism of the car was implemented using steering principles, which involved calculating the turning radius based on the steering angle and the car's chassis length. To simulate the turning motion of the car, we employed steering principles based on the geometry of circular motion. When the front wheels are steered, the car follows a circular path, with the radius of this circle determined by the length of the car chassis divided by the sine of the steering angle. This calculation enables us to determine the turning radius and, consequently, the curvature of the car's trajectory during turns.

Car Physics: The car's position is updated based on its velocity, accounting for changes in speed and direction over time. Additionally, gravity is applied to simulate the vertical movement of the car, ensuring realistic behavior when driving on inclined surfaces.

Camera Position: The camera position is dynamically updated to track the car's movement, providing a realistic viewpoint for the user. By adjusting the camera's position relative to the car, we create an immersive experience that enhances the sense of presence within the simulation.

Differential Equations for Car Dynamics

1. **Translation Dynamics:**

$$\frac{dx}{dt} = vcos\theta$$
$$\frac{dz}{dt} = vsin\theta$$

2. **Rotation Dynamics:**

$$\frac{d\theta}{dt} = \omega$$

3. **Speed Dynamics:**

$$\frac{dv}{dt} = a - \mu \cdot v$$

4. **Steering Dynamics:**

$$\frac{d\dot{\theta}}{dt} = \gamma \cdot v - k \cdot \dot{\theta}$$

5. **Ground Collision:**

$$\frac{dv_y}{dt} = g - r$$

where:

v is the car's speed
v_y is the vertical velocity
θ is the car's rotation angle
$\dot{\theta}$ is the rotation speed
a_b is the acceleration due to pressing the accelerator
a_f is the deceleration due to friction
γ is the steering amount
k is a damping factor
g is the gravitational acceleration
r is the reaction force from the ground
t is the time duration of the simulation

3 Experiments, implementation

We organized our car simulation project into a single main component, represented by the ”carsim.py” file. This file acts as the central hub for the simulation, leveraging the Ursina framework. It handles the loading of essential models for both the car and the racetrack, enriching the visual aspect of the simulation. Within ”carsim.py,” the Car class encapsulates the behavior and physics of the car entity. This class manages crucial aspects such as acceleration, braking, steering, and various realistic physics parameters like velocity, acceleration, and gravity. The track object file and texture were created using blender.

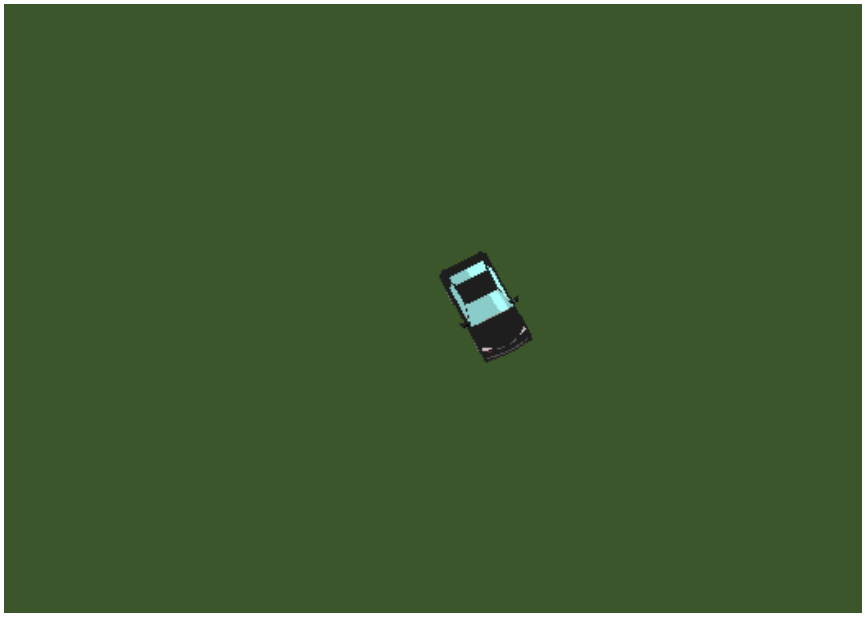
The update loop continuously adjusts the car's position, rotation, and physics parameters based on user input and environmental conditions. Additionally, camera functionality is implemented to provide dynamic viewpoints, enhancing the overall user experience.

By consolidating the Car class within ”carsim.py,” we maintain a streamlined and organized codebase, facilitating easier maintenance, debugging, and expansion of the car simulation project. Leveraging the Ursina framework enables us to create visually appealing and interactive simulations with advanced features such as drifting, camera manipulation, and collision detection.

4 Results

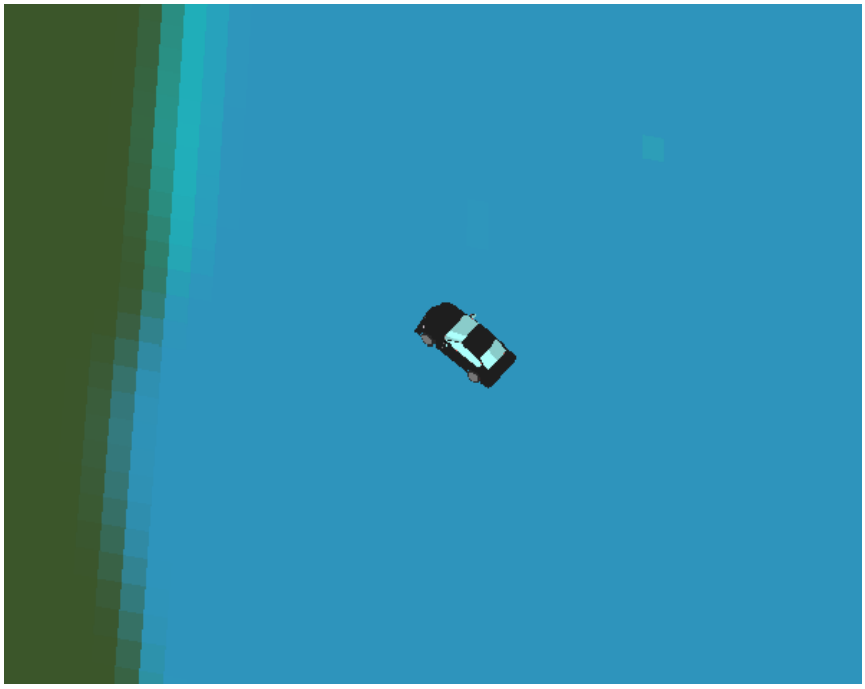
To set up the different terrains, we defined different environmental terrains such as water, mud, and sand to add variety and challenges to the simulation. For each terrain type, we specified coordinates that define the boundaries of the terrain area. These coordinates represent the outer edges of the terrain pads, forming square grids. By extracting the x and z values from these coordinates, we created a square grid to determine if the car's position falls within the boundaries of each terrain type. To check if the car is in water, we compare its x and z coordinates with the predefined water pad coordinates. If the car's position lies within these boundaries, it is considered to be in water. Similar checks are performed for mud and sand terrains.

The green represents grass which is free to drive on without any inflicting conditions

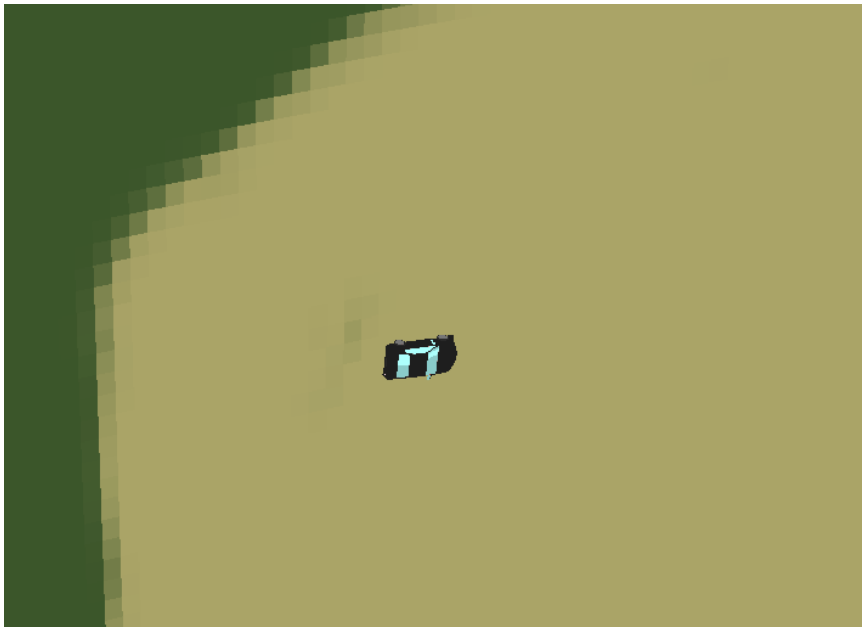


The brown represents mud and conditions are slippery when the car is on that texture part.

The blue represents water and the car will not be able to drive comfortably on this texture.



The tan represents sand and conditions are slippery and car's acceleration is drastically slowed.



The grey represents rocks/boundaries of the track. When close or touching the game will ask the user to reset the car.

We implemented the boundaries by utilized collision detection to define boundaries for the car within the simulation. If the car intersects with any entity, its position is reverted to prevent it from going beyond the defined boundaries. A reset prompt is displayed if the car intersects with any entity, and a timer is initiated to hide the prompt after a certain duration.

Additionally, we implemented gravity to simulate the car's movement when it's not on the ground. If the car's position exceeds a certain threshold indicating it's not on the ground, gravity is applied to bring the car back down. This threshold ensures that the movement remains smooth and that the car stays close enough to the ground, preventing violent shaking when sliding on the track.

5 Conclusions

To conclude, our car simulation project showcases a semi-realistic driving experience by incorporating real-world physics principles for car movement. The implementation of steering, acceleration, and braking mechanisms closely mimics real-life driving dynamics, contributing to an immersive simulation environment. The simulation accurately replicates advanced driving techniques such as counter-steering, which facilitates realistic drifting maneuvers you would see from an actual vehicle.

The car simulation in our project may technically represent a box rather than a true rigid body, the manipulation of forces on this box allows for accurate simulation of car dynamics. The tracks, although not perfectly smooth, required careful adjustments to ensure smooth movement of the car. Incorporating a threshold for car sliding on the track helped maintain stability and proximity to the ground, enhancing the overall driving experience.

A References

[1] **3D Model of Car:**
Poly Pizza. (n.d.). Retrieved from `https://poly.pizza/m/dVLJ5CjB0h`

[2] **Track Creation:**
Blender Foundation. (n.d.). Blender. Retrieved from `https://www.blender.org/`

[3] **Car Turning/Steering Tutorial:**
RMGI. (n.d.). Pygame 2D Car Tutorial. Retrieved from `https://rmgi.blog/pygame-2d-car-tutorial.html`

[4] **Framework Development Video:**
Mandaw. (2022, March 17). Developing a Game Framework in Python with Pygame [Video]. YouTube. Retrieved from `https://www.youtube.com/watch?v=VBYanr-lpoE&ab_channel=Mandaw`