Final Project: Police Car Chase

Team 12: Khoi Nguyen, Kevin Yi

CS 174A: Demetri Terzopolous

Overview

Our final project is a police car chase game. In it, you play a cop driving a police car around a small city area. Each human roaming around the city is a criminal, which you have to hit with your car in order to arrest them. Once every single human is arrested, you win the game. In terms of advanced features, our project utilizes simple collision detection, simple driving physics, and a hierarchical object (police car).

Methods

Cop Car:

The cop car is a hierarchical object that the player controls using the keyboard. To draw the actual cop car, a matrix, cop_car, is computed to represent the car's position for that tick. The cop car's state is represented by class variables, as described in the following table:

Variable Name	**Description**
this.cop_x/y/z	Represents the cop's position
this.cop_cam	Stores the camera matrix for 3rd person view behind cop
this.move	Boolean showing if the cop car is currently moving. True if forward/backward keys are pressed, false otherwise.
this.move_direction	1 if car is moving forwards (forward button is held) -1 if car is moving backwards (back button is held)
this.turn_left/right	Boolean showing if left/right buttons are currently held. If true, rotates wheels left/right. If false, wheels stay in the same place.
this.cop_front_rotation	Represents the angle of rotation of the wheels relative to the cop car
this.cop_angle	Represents the angle of rotation of the cop car relative to the scene

Each tick, before the cop car is drawn, the following steps are taken to calculate the proper matrices:

1. Create a new matrix representing the cop car's current position.
2. Change this.cop_front_rotation based on whether the turn_left or turn_right buttons are held.
3. If the cop car is moving, then calculate the new x and z components of the cop car, based on turning angle and set velocity. Also recalculate the x and z positions of the camera.
4. Rotate cop car based on new this.cop_angle.
5. Calculate camera matrix based on new cop position and new camera position.

This matrix, along with the cop velocity, are both passed into the drawCopCar() function. Within the drawCopCar() function, drawing is split into 6 different parts: car body, windshields, windows, top of cop car, front wheels, and back wheels. Most of the function is self-explanatory other than the wheels, which are a little more complex. Each wheel rotates either forwards or backwards depending on the car's direction. The front wheels also pivot based on the angle in this.cop_front_rotation.

To detect collisions against the car, only the point at the middle of the car is used. This makes it so that there is some leniency when hitting buildings, but we found that it does not make it much more difficult to hit the humans.

Humans:

Each human is composed of 5 different cubes for the body and a sphere for the head. To draw a single human, the function drawHuman() is called. Given a position and a color, the function uses the position as a new coordinate system and draws each of the shapes, with the shirt being the color of what is passed in. Because there are 10 humans at the start of the game, the following variables represent each of the humans' states:

Variable Name	**Description**
this.human_pos	A two dimensional array that stores the x and z coordinates for every human. The y coordinate is always 0, thus is not stored.
this.new_transform	A two dimensional array that stores the direction of where the human should move.
this.colorArr	A two dimensional array that stores the color shirt of each human. The shirt colors are generated randomly and will change throughout the game.
this.aliveHumans	An array that stores whether each human is alive or not.

The humans are designed to move randomly in order to make the game difficult. Every two seconds, each human decides on a direction with a random x and z distance, ranging from [-1,1), from the human's position. During each tick of the two second interval, the humans first checks if it has collided with the cop car. It checks this by comparing the distance between the car and the human and if the distance is less than a certain threshold. The threshold is calculated by adding the distance from a human's origin to the furthest point contained by the human to the distance from the cop car's origin to the furthest point contained by the cop car. If a collision

with the car is detected then the aliveHumans element corresponding to that human is changed to false.

After checking collision with the cop car, the next step during the tick is to decide the next position to move each human. To get the next position, we add the direction to the current position. Before the new position is passed into the drawHuman function, we check if the position is located within a building's bounding box or if the location already matches another human. If it does, then the human keeps choosing a new direction until the new position is not within a building's bounding box nor another human. Once a valid location is chosen, the human_pos array is updated. Once all the human positions are updated, we then call drawHuman on only alive humans.

Buildings:

In the game we have two buildings, constructed from texture mapped cubes. With the buildings, it is important to calculate the bounding box of each building to know when there is collision and when the humans need to change direction. To calculate the coordinates of the bounding box, we take the matrix of the original cube and multiply it by translation and scale matrix for each building. After that we expand the box by 1 unit in the x and z direction so that the humans do not enter the building before changing directions.

Game Logic:

The game logic is mostly contained within an if-statement at the beginning of the display() function. The game starts at the title screen, which is a square texture located far away from the playing field. Once 'b' is pressed, then the game starts, transporting the camera to the

cop car. The function, hit_building(), checks the player's current position against the building coordinates and the playing field bounds. If the player's position collides with a building or goes out of bounds, then the camera transports to the game over screen. The function, check_win(), scans through the array this.aliveHumans to see if all the humans are dead. If all the humans have been arrested, then the completion screen is displayed.