CS616 Project Report

Matthew Moore

April 29, 2018

My original goal was as follows: create a program using C++ and OpenGL that creates a 3D simulation of the movement of a vehicle using rigid bodies and simple geometric shapes.

- The user controls a rigid body by accelerating and turning.
- The viewpoint of the user will be at a fixed position relative to the rigid body they control.
- The movement of rigid bodies will be affected by collision (collision detection).

As of this draft, all the above goals have been implemented and well-tested. Additionally, I have added a camera that looks down on the position of the user-controlled rigid body. The user can alternate their current view between this camera and the abovementioned camera. The movement of rigid bodies and the camera are based on real time. A collision of one body with another multiplies the velocity of the former body by a negative factor. A collision of one body with another does not otherwise affect the other body.

Through implementing this project, I have gained experience in the following areas:

- Planning a software model for a 3D coordinate system
- 3D rotation and angle calculations
- GL projection view and model view
- Synchronizing 3D movements with real time
- Calculation of collision with respect to cuboids

The classes I used in my implementation are on the following pages.

Object:

- This represents a generic 3D object in space.
- Fields:
  - Position vector
  - Normalized basis vectors
  - Horizontal and vertical angles
    - Horiontal angle is measured from the vector {0, 0, 1}
    - Vertical angle is measured from the vector {0, 0, 1} rotated by the horizontal angle
  - A 3x3 matrix used for rotation
- Methods:
  - Movement by the basis vectors
  - Translation to coordinates or by offset
  - Horizontal or vertical rotation by an angle, to an angle, or to an angle offset of a vector

Camera (inherits Object):

- This represents a camera in 3D space.
- Fields:
  - A point in space representing the target of the camera
  - A normalized basis vector that determines the roll of the camera
  - The roll angle of the camera (the angle from the relative Y basis vector to the roll vector)
  - The distance to the target
- Methods:
  - Movement by the basis vectors
  - Translation to coordinates or by offset
  - Rotation by an angle, to an angle, or to an angle offset of a vector
  - Revolution around the target by an angle, to an angle, or to an angle offset of a vector
  - Setting of the target point and subsequent rotation to face this point
  - Movement toward the target by a certain distance
  - Movement to a certain distance from the target

Shape (inherits Object):

- This represents a drawable 3D object with a basic shape (only cuboids in my implementation).
- Fields:
  - A code representing the type of shape (e.g. cuboid)
  - Scale factors that determine the size
  - Color, reflectance, and shininess which determine the texture
- Methods:
  - Setting of the scale factors either individually or collectively
  - Setting of the color, reflectance, or shininess
  - Testing whether a point is inside the shape (only cuboid)
  - Testing whether another shape (only cuboid) is inside the shape (only cuboid)
    - This uses the Separating Axis Theorem (SAT)
  - Drawing the shape within the GL display loop using GLUT functions

SAT explanation: <https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem–gamedev-169>

Acceleration:

- Keeps track of a one-dimensional velocity affected by acceleration and friction.
- Fields:
  - The current velocity
  - The current acceleration
  - A friction factor that is removed from the velocity at each time step
  - The minimum velocity if the current acceleration is 0
- Methods:
  - Setting of current velocity
  - Setting of the current acceleration
  - Setting of the friction factor
  - Setting of the minimum velocity
  - Updating the current velocity based on an input time step and the current fields

The algorithm of my implementation is as follows:

- Display the valid input commands to the console
- Initialize frame rate clock
- Create GL window
- Set display method and input methods that GL will use
- Enable GL rendering functions
- Initialize cameras and shapes
- Start GL display loop
  - Exit with certain input flag
  - Update frame rate values
  - Perform functions based on input flags
  - Update shape positions based on acceleration
  - Update camera positions
  - Update the dimensions of the GL window
  - Set the GL light position and intensity
  - Set the GL projection to perspective
  - Set the background color
  - Set the view position based on the current camera
  - Draw the shapes
  - Display the scene

The result of my implementation is a playable simulation of the basic 3D movements of driving. Working on this project required great amounts of precision and planning, but I feel it was worth the effort. Overall, I enjoyed working on this project, and I am generally satisfied with the current results. The only changes I would make at this point would be to add more collideable bodies to the implementation in order to provide a better user experience, or to improve the rendered scene by adding more features, such as shadows.