Lecture 9

Acceleration
Deceleration
Friction

CSD1130
Game
Implementation
Techniques

1. ASTEROIDS – Physics

1.1. Velocity – Moving Along Vectors

- In most games, sprite movement is done according to some vector and speed.
- The direction of the vector is usually determined by user input, collision reaction and reflection.
- The movement speed is determined by a "speed" variable and not by the direction vector's length.
- **Velocity = Normalized Direction * speed**
- Moving a sprite is done by changing its current position (which will later result in a new translation matrix).
- Getting the next position of a sprite whose speed is S and direction vector is V:
- Next Position = S*V*TimeStep + Current Position
    - The mathematical formula of the above equation is:
        - $Pos = V*t + Pos_0$. The only difference is that in real time simulation, Pos_0 represents the last frame's position.

- Example:
    - T is a 2D point located at (0;0)
    - Direction vector is (0.31 ; 0.95) (Normalized, explained later).
    - Its speed value is 2.
    - Assume the TimeStep is 1.0 second (In games, the TimeStep is the frame's frame time).

    - Frame 0:
        - X0 = 0
        - Y0 = 0

    - Frame 1:
        - X1 = Direction(X)*speed*ts + X0 = 0.31*2*1 + 0 = 0.62
        - Y1 = Direction(Y)*speed*ts + Y0 = 0.95*2*1 + 0 = 1.9

    - Frame 2:
        - X2 = Direction(X)*speed*ts+ X1 = 0.31*2*1 + 0.62 = 1.24
        - Y2 = Direction(Y)*speed*ts+ Y1 = 0.95*2*1 + 1.9 = 3.8

    - Frame 3:
        - X3 = Direction(X)*speed*ts + X2 = 0.31*2*1 + 1.24 = 1.86
        - Y3 = Direction(Y)*speed*ts + Y2 = 0.95*2*1 + 3.8 = 5.7

- As long as the direction vector isn't changed, the sprite will keep moving in the same direction (Unless its speed is set to 0).
- On the other hand, if the speed's value goes from being positive to negative, the sprite direction will become the opposite of its own direction vector.
- Note that in most games, the direction vector is normalized, and the object's movement speed is determined by the "speed" value.
- This normalization, which separates the object's direction from its speed, allows us to move an object in any direction at a constant speed.
  - The direction vector's only responsibility is to direct the object.
  - And the object's speed is controlled only by the speed value.
  - Keeping the speed value "2" for example will ensure that that particular object will move 2 units per second no matter what its direction vector is (as long as that direction vector is normalized).

1.2. Acceleration

- Object's velocity doesn't have to be constant.
- In can be altered by adding the acceleration element.
- The acceleration affects the velocity the same way the velocity affects the position.
- Getting the next position of an object whose velocity is V and acceleration is A:
  - Next Position = ½ A*TimeStep*TimeStep + V*TimeStep + Current Position
    - The mathematical formulas of the above equation are:
      - $Pos_1 = ½A*t*t + V_0*t + Pos_0$.

      The only difference is that in real time simulation, Pos_0 represents last frame's position, V_0 represents last frame's velocity and Pos_1 represents the current frame's position.
      - $V_1 = A*t + V_0$. V_1 is the current frame's velocity and V_0 is last frame's velocity.

  - Note that the speed value (used previously) doesn't exist anymore. This is because the change in velocity is affected by the acceleration, and not by a hard-coded "speed" value.

1.3. Acceleration – In Games (example Asteroids)

- Object's velocity doesn't have to be constant in games.
- In can be altered by adding the acceleration element.
- The acceleration affects the velocity the same way the velocity affects the position.
- In games, the player usually can control the acceleration of objects, which will implicitly change the objects' velocity and eventually position.
- Within one game loop (or one frame) the velocity is considered constant, and no acceleration is applied.
    - The acceleration will affect/change the velocity from one game loop to another, and not within/inside a game loop.
- Getting the next position of a sprite whose next velocity V_1 is calculated based on the previous velocity V_0, and acceleration is A:
    - Next Velocity = V_1 = A*TimeStep + V_0
    - Next Position = V_1*TimeStep + Current Position
        - The mathematical formulas of the above equations are:
            - $V_1 = A*t + V_0$. V_1 is the current frame's velocity and V_0 is last frame's velocity.
            - $Pos_1 = V_1*t + Pos_0$.

            The only difference is that in real time simulation, Pos_0 represents last frame's position, V_0 represents last frame's velocity and Pos_1 represents the current frame's position.

- Example:
    - T is a 2D point located at (0;0),
    - Velocity is (0, 0) (The object is initially not moving)
    - Its acceleration is (3, 2) during frame 1 and frame 2, then it goes back to (0, 0) in frame 3. (This can be the result of the player pressing the “forward” button during frame 1 and 2)
    - Assume the TimeStep is 1.0 second.

- Frame 0:
  - $X0 = 0$
  - $Y0 = 0$

- Frame 1:
  - $V1_x = A_x*ts + V0_x = 3*1 + 0 = 3$
  - $V1_y = A_y*ts + V0_y = 2*1 + 0 = 2$

  - $X1 = V1_x*ts + X0 = 3*1 + 0 = 3$
  - $Y1 = V1_y*ts + Y0 = 2*1 + 0 = 2$

- Frame 2:
  - $V2_x = A_x*ts + V1_x = 3*1 + 3 = 6$
  - $V2_y = A_y*ts + V1_y = 2*1 + 2 = 4$

  - $X2 = V2_x*ts + X1 = 6*1 + 3 = 9$
  - $Y2 = V2_y*ts + Y1 = 4*1 + 2 = 6$

- Frame 3:
  - $V3_x = A_x*ts + V2_x = 0*1 + 6 = 6$
  - $V3_y = A_y*ts + V2_y = 0*1 + 4 = 4$

  - $X3 = V2_x*ts + X2 = 6*1 + 9 = 15$
  - $Y3 = V2_y*ts + Y2 = 4*1 + 6 = 10$