Complex Motion

Complex Motion

Motion means that the position changes over time.

We'll consider two patterns of how the position is changed –

- by updating the position
  - add or subtract some amount from the previous *x*, *y* to get the new *x*, *y*
- by computing the position outright
  - have some formula to compute *x*, *y* directly

These ideas are not limited to motion – any value (size, color, etc) can be updated/computed in the same ways.

Updating Position

new position = old position + speed × time interval

- we usually take the time interval to be one frame so this simplifies to

  new position = old position + speed

- or, when updating animation variables

  ```
  position = position + speed
  ```

  - instead of adding or subtracting depending on the direction of movement, treat speed as positive or negative
    - positive → moving right (`xspeed`) or down (`yspeed`)
    - negative → moving left (`xspeed`) or up (`yspeed`)

  which is implemented as

  ```
  x = x + xspeed
  y = y + yspeed
  ```


Steady Motion

```
position = position + speed
```

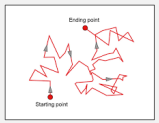
- for steady motion, the speed is constant so we typically don't bother with variables for speed (e.g. `x = x+1`)

Random Walk

position = position + speed

- the speed doesn't have to be constant

- for a *random walk*, use a small random number for the speed
  - `random(8)` – a `float` between 0 and 8, but not including 8 itself
  - `random(-5,10)` – a `float` between -5 and 10, but not including 10 itself
  - `int(random(-5,10))` – an `int` between -5 and 10, but not including 10 itself
- random walks are a good simulation of *Brownian motion*, the random movement of particles in a liquid or gas as they get bumped by individual molecules or atoms



https://www.fritsch-international.com/particle-sizing/fritsch-knowledge/brownian-motion/

Speeding Up and Slowing Down

position = position + speed

- the speed can itself change over time

What changes? position *and* speed
- need animation variables for both

- *acceleration* is the rate of change of speed
  - often constant, so no need for an acceleration animation variable
  - can be in the same direction as speed → speeding up
    - same sign
  - can be in the opposite direction of speed → slowing down (deceleration)
    - opposite sign

position = position + speed
speed = speed + acceleration

Physically-Based Motion Simulation

position = position + speed
speed = speed + acceleration

How do we change the speed in a realistic way?
Elements –

- *gravity* is acceleration downwards
  - only affects `yspeed`
  - positive number because positive speed means moving downwards
    - `yspeed = yspeed + g` (g > 0)
- *air resistance* is deceleration proportional to speed
  - amount depends on speed, in opposite direction
    - `speed = speed – k*speed` (k ≥ 0, k << 1)

<< means "much less than"
>> means "much greater than"

Physically-Based Motion Simulation

position = position + speed
speed = speed + acceleration

Additional elements –

- *bouncing* is changing direction to go the other way when a surface is hit
  - sometimes bounce, sometimes don't means …?
    - on-the-spot "to do or not to do" – `if ( have hit ) { do the bounce }`
  - "going the other way" = change the sign of `xspeed` when a vertical wall is hit, change the sign of `yspeed` when a horizontal wall is hit
    - `speed = -1*speed`
- *damping* is energy lost due to friction when bouncing
  - object doesn't bounce as high each time
  - `speed = -k*speed` instead of `speed = -1*speed` (k < 1, k >> 0)

<< means "much less than"
>> means "much greater than"