Principles of Computer Game Design and Implementation

Lecture 15
We already learned

• Collision Detection
  – two approaches (overlap test, intersection test)
  – Low-level, mid-level, and high-level view
Collision Response

• What happens after a collision is detected?
  1. Prologue
     • Check if collision should be ignored
     • Sound / visual effects
  2. Collision
     • *Resolve collision*
  3. Epilogue
     • Propagate the effects
       – destroy object(s), play sound...
Collision Resolution

- **Animation based**
  - An artist models collision
    - A rocket hits a target...
  - Motion-capture
    - Sport games

- **Physics based**
  - Generated by an algorithm
  - Based on (more or less) realistic models
Recall: Classic Game Structure

- A convexity
- Starts with a single choice, widens to many choices, returns to a single choice
Why Physics?

• Responsive behaviour
  – Infinitely many possibilities

• For centauries people were *describing* the world
  – We can use the equations to *model* the world

• Can be hard
  – Knowledge of physics
  – “Real” physics is too expensive computationally
“Motion Science” in Games

• Kinematics
  – Motion of bodies without considering forces, friction, acceleration,…
  – Not realistic

• Dynamics
  – Interaction with forces and torques
Separate translation and rotation

• Particle physics
  – A sphere with a perfect smooth, frictionless surface. No rotation
  – Interaction with forces and environment
    • Position, Velocity, Acceleration

• Solid body physics
  – Torques, angular velocity, angular momentum
Continuous Motion

• Particles move in a “smooth way”
  – Position as a function of time
    \[ P(t) \] is the position of \( P \) in the moment \( t \)
  – The derivative
    \[ \frac{dP(t)}{dt} \]
    describes how \( P(t) \) changes over time
• Velocity (speed)
Discrete Particle Motion

• Uniform motion
  – Nothing affects the motion

• Gravitational pull
Integrators

• The process of computing the position of a body based on forces and interaction with other bodies is called *integration*.

• A program that computes it is an *integrator*.
Newton’s Laws

1. Every body remains in a state of rest or uniform motion unless it is acted on by an external force

2. A body of mass \( m \) subject to force \( F \) accelerates as described by
   \[ F = ma \]

3. Every action has an equal and opposite reaction
Position and Velocity

Continuous physics

- \( \mathbf{V}(t) = \frac{d\mathbf{P}(t)}{dt} \)

- \( \mathbf{P}(t) = \ldots \text{ (maths)} \)

Discrete physics

- \( \mathbf{V}(t) = \frac{\Delta \mathbf{P}(t)}{\Delta t} = \frac{\mathbf{P}_{i+1} - \mathbf{P}_i}{tpf} \)

- \( \mathbf{P}_{i+1} = \mathbf{P}_i + tpf \cdot \mathbf{V}(t) \)

Main loop iteration
Time per frame
Recall: Arbitrary Translation

- Every iteration *update* the position

\[ P = P + speed \cdot tpf \cdot U(t) \]

- \( U(t) \) - the direction of movement
  - Depends on time!!
- \( speed \) is speed
- \( tpf \) is time per frame
Velocity and Acceleration

Continuous physics

- \[ a(t) = \frac{dV(t)}{dt} \]
- \[ V(t) = \ldots \text{(maths)} \]

Discrete physics

- \[ a(t) = \frac{\Delta V(t)}{\Delta t} = \frac{V_{i+1} - V_i}{tpf} \]
- \[ V_{i+1} = V_i + tpf \cdot a(t) \]

Main loop iteration

Time per frame
Example: Gravitational Pull

- \( \mathbf{a}(t) = \mathbf{g} = 9.8 \text{ N/kg} \)
- \( \mathbf{V}_{i+1} = \mathbf{V}_i + tpf \cdot \mathbf{g} \)
- \( \mathbf{P}_{i+1} = \mathbf{P}_i + tpf \cdot \mathbf{V}_{i+1} \)

Vector3f velocity = new Vector3f(10, 10, 0);
Vector3f gravity = new Vector3f(0, -9.8f, 0);
...
public void simpleUpdate() {
    velocity = velocity.add(gravity(tpf));
    ag.move(velocity.mult(tpf));
}
Acceleration and Force

Newton’s second law: a body of mass $m$ subject to force $F$ accelerates as described by

$$F(t) = ma(t)$$

$$a(t) = F(t)/m$$

Example:

- Engine thrust $F_{\text{engine}} = kU_V$
- Linear drag $F_D(t) = -bV(t)$
- Quadratic drag $F_{QD}(t) = -c|V(t)|^2V(t)$
Example: Pull + Drag

\[
F_{i+1} = -bV_i \\
a_{i+1} = g + F_{i+1}/m \\
V_{i+1} = V_i + tpf \cdot a_{i+1} \\
P_{i+1} = P_i + tpf \cdot V_{i+1}
\]

Vector3f force = velocityB.mult(-b);  
accelerationB = gravity.add(force.divide(m));  
velocityB = velocityB.add(accelerationB.mult(tpf));  
bg.move(velocityB.mult(tpf));
Example: Pull + Drag + Thrust

\[ F_{i+1} = -bV_i + kU \]  
\[ a_{i+1} = g + F_{i+1}/m \]  
\[ V_{i+1} = V_i + tpf \cdot a_{i+1} \]  
\[ P_{i+1} = P_i + tpf \cdot V_{i+1} \]

Vector3f directionC = velocityC.normalize();  
Vector3f forceC = velocityC.mult(-b).add(directionC.mult(thrust));  
accelerationC = gravity.add(forceC.divide(m));  
velocityC = velocityC.add(accelerationC.mult(tpf));  
cg.move(velocityC.mult(tpf));
Simulation Recipe

• Add up all the forces acting on the object
  – Gravity, drag, thrust, spring pull,…
• Represent the motion as discrete steps

\[
\begin{align*}
a_{i+1} &= g + \frac{F_{i+1}}{m} \\
V_{i+1} &= V_i + tpf \cdot a_{i+1} \\
P_{i+1} &= P_i + tpf \cdot V_{i+1}
\end{align*}
\]

Euler steps
Rotation

• Rotation of a uniform (again simplification) solid body can be described mathematically
  – Speed vs angular speed
  – Force vs torque
• Represent as discrete motion
• Use Euler steps to compute the rotation matrix
• Combine with translation

Hard but doable
Accuracy of Simulation

• How accurate this simulation is?

• Does it matter?
  – It’s all about illusion, if the behaviour looks right, we do not care.

• But...