Game Physics

CSD2401

Physically Based Motion

- Want game objects to move consistent with real world expectations
- Player expectations include movie physics
- Needs to be stable (floating point errors!)
- Needs to be deterministic
- Needs to be fast

Simple Physics

Simulate simple Newtonian rigid body dynamics

Basic equations of motion

Objects are simple convex hulls with uniform density

No molecular, atomic, or relativity-based calculations

Continuous Physics

Search for collision in continuous time and find expected time of impact for each object pair

Determine the first colliding pair, move forward to that time and resolve the collision.

Problems

- Expensive and complex

Discrete Physics

Objects are moved with basic Newtonian physics by a time interval (a frame)

Collisions are detected at sample intervals in time.

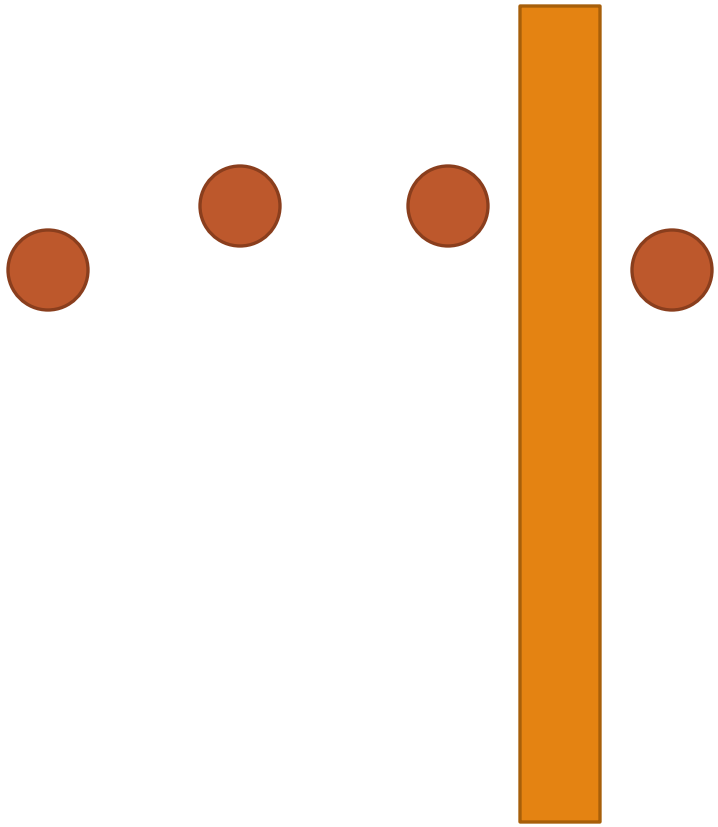
Inaccurate but realistic looking

Very fast

Problems

- Objects can tunnel (miss samples)
- Collisions are detected too late (objects are interpenetrating)

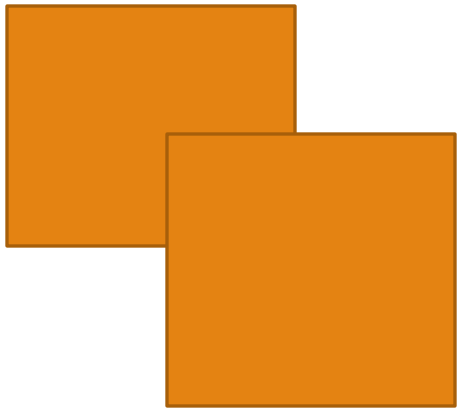
Discrete Problems - Tunneling



Discrete Problems – Late Detection

What is the collision normal?

How did these objects get this way?



Discrete Problems - Solutions

Move objects in small steps within a frame

- Expensive

Increase collision data size

- i.e. Bounding Rectangle

Favor Discrete

In cases where you don’t need accuracy

Much faster and simpler then Continuous

Result is good enough

Can be scaled to higher framerate

Dynamics

Linear Dynamics

Have:

- Last frame position $\mathbf{x}_i$
- Last frame velocity $\mathbf{v}_i$
- Mass m
- Set of forces $\mathbf{F}$

Need to determine:

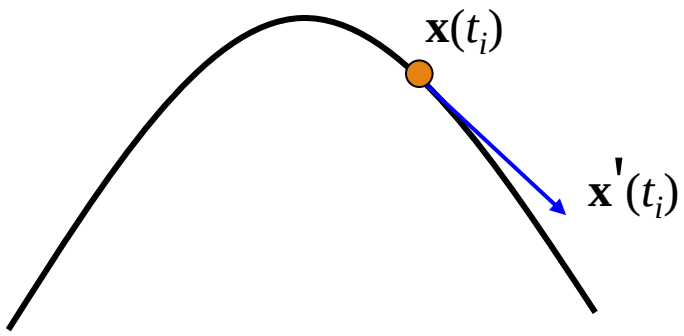
- Current frame position $\mathbf{x}_{i+1}$
- Current frame velocity $\mathbf{v}_{i+1}$

Differential Calculus

Have position function $\mathbf{x}(t)$

Derivative $\mathbf{x}'(t)$ describes how $\mathbf{x}$ changes as t changes (also written as $d\mathbf{x}/dt$)

$\mathbf{x}'(t)$ is tangent vector at time t



Differential Calculus

Our function is position: $\mathbf{x}(t)$

Derivative is velocity: $\mathbf{v}(t) = \mathbf{x}'(t) = \frac{d\mathbf{x}}{dt} = \dot{\mathbf{x}}$

Derivative of velocity is acceleration: $\mathbf{a}(t) = \mathbf{v}'(t) = \frac{d\mathbf{v}}{dt} = \dot{\mathbf{v}}$

$= \mathbf{x}''(t) = \frac{d^2\mathbf{x}}{dt^2} = \ddot{\mathbf{x}}$

Newtonian Dynamics

All objects affected by forces

- Gravity (constant acceleration)
- Ground Normal Force
- Other objects pushing against it

Force determines acceleration ($\mathbf{F} = m\mathbf{a}$)

Acceleration changes velocity ($\frac{d\mathbf{v}}{dt} = \mathbf{a}$)

Velocity changes position ($\frac{d\mathbf{x}}{dt} = \mathbf{v}$)

Summing Forces

D’Alembert’s Principle

$$\mathbf{F} = \sum_k \mathbf{F}_k$$

All forces on a body can be summed into a single force.

Simple, just add all forces together

Linear Dynamics

Assume force is constant for a frame

Use velocity to take a small step forward by frame time dt

Repeat for velocity and acceleration

This is Euler’s method

Linear Dynamics

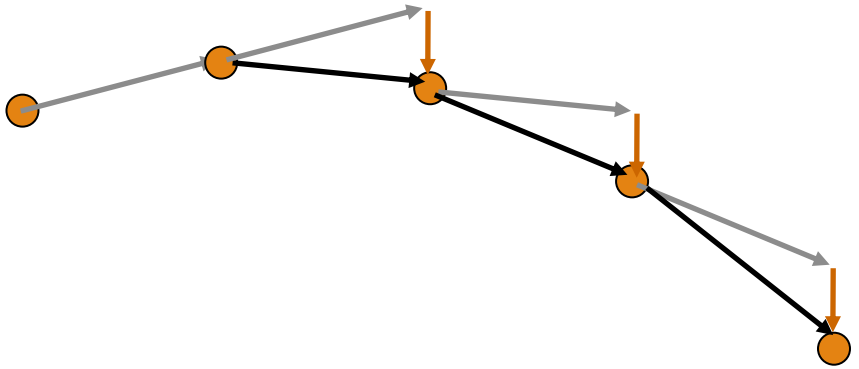
Euler Integration with time step dt

$$\mathbf{F} = \sum_k \mathbf{F}_k$$

$$\mathbf{a}_i = \frac{\mathbf{F}}{m}$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + dt\mathbf{a}_i$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + dt\mathbf{v}_{i+1}$$



Simple Linear Dynamics

```
void Body::Integrate(float dt)

{

        //Determine the acceleration

        Acceleration = SummedForce * InvMass + Gravity;


        //Integrate the velocity

        Velocity = Velocity + Acceleration * dt;


        //Update position

        Position = Position + Velocity * dt;

}
```

Forces

Forces – Types

Linear Directional Force

Rotational Force

Mixed

Drag Force

Forces – Types

Linear Directional Force – Props

- Unit direction
- Magnitude
- Lifetime
- Age (current time since created/activated)
- IsActive

Forces – Types

Rotational Force – Props

- Torque
- Lifetime
- Age (current time since created/activated)
- IsActive

Forces – Types

Drag Force – Props

- Directional Drag (a scalar: applies opposite of linear velocity)
- Rotational Drag (a scalar: applies opposite of angular velocity)
- Lifetime (always)
- IsActive

Force – Usage

General Calls

```
{
        //Some Useful Calls
        AddForce(…); //for each type (if all under one class)
        RemoveForce();
        ActivateForce(bool);
        ValidateAge();
        SetLifeTime();
        …
}
```

Forces Manager

It is a very useful design/structure to have a ForcesManager class

It can validate all applied forces on an entity (GOC) and returns the sum resultant force

Dynamics – Component

"Dynamics" can be a component that holds the physics dynamics for an entity.

Possible properties

- Mass
- Inertia Mass //if rotational physics
- Position
- Velocity and Acceleration
- Angular Velocity and Angular Acceleration //if rotational physics
- Center of Mass
- ForcesManager (an instance)

Physics – Update

Directional Form

- Sum(Forces) = (mass) * (acceleration)
  - (acceleration) -> Integrates -> (velocity)
  - (velocity) -> Integrates -> (position)

Rotational Form

- Sum(Torques) = (inertia mass) * (angular acceleration)
  - (angular acceleration) -> Integrates -> (angular velocity)
  - (angular velocity) -> Integrates -> (angle)

Physics – Update

Important Notes

- Physics position must overload the transformation's translation
- Physics angle must overload the transformation's orientation (or rotation)
- OR vice versa!

Collision

Collision Detection

Every object against every other is slow $O(n^2)$

Broad-Phase finds objects near each other

Narrow-Phase checks on these pairs

Collision Detection

Broad Phase: Determine all possible collision pairs

- Grids
- Trees
- Sweep and Prune

Narrow Phase: Determine contacts between two primitive shapes

- Separating Axis Theorem
- GJK

Resources

Game Physics Engine Development

Erin Catto's Weblog

JigLib

References

Real-Time Collision Detection – Ericson

Dynamics Slides from GDC presentation by Jim Van Verth

Game Physics Engine Development – Ian Millington

FPS vs FixedDT

UPLOADED ON MOODLE: FPS-FIXEDDTIME.PDF

Real Time Demo

SHOWN DURING LECTURE TIME