Fuerzas

A. Solana
D. Garcés
M. Guijarro
L. García
M. Gómez

Combinación

- Hasta ahora una fuerza → Gravedad
- Normalmente varias fuerzas actúan
  - Fuerzas de campos
  - Fuerzas de restauración
  - Fuerzas de fricción
  - Fuerzas viscosas

- Se pueden combinar en una resultante
- La fuerza resultante es la suma de todas las fuerzas

$$f = \sum_i f_i$$

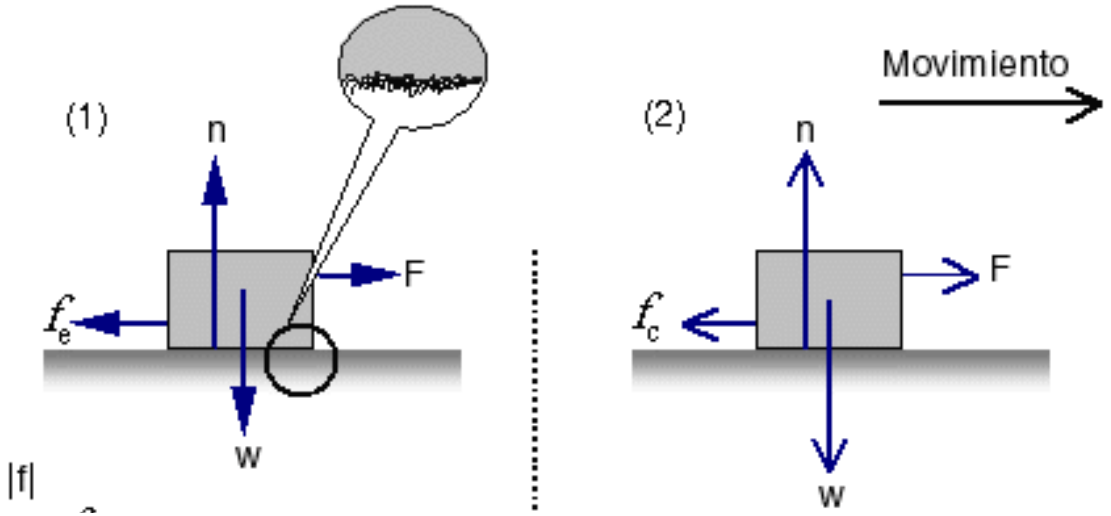
- Es necesario modelar de una forma sencilla las fuerzas que aparezca en nuestro sistema:
  - Fuerzas de campos
  - Fuerzas de restauración
  - Fuerzas de fricción
  - Fuerzas viscosas

- Generadas por campos gravitacionales, eléctricos, magnéticos,…
  - Formulación matemática que depende de:
    - Posición, velocidad
    - Masa, carga, dipolo, …
    - Ctes. universales
  - Versiones Simplificadas
    - R_e=6.38*10^6m y M_e=5.98*10^{24}kg
    - Para *h=0* obtenemos que $F_g/m=g$=9.8m/s^2

$$F_a = (G \cdot m_1 \cdot m_2) / r^2$$

$$G=6.673*10\text{-}11\ (N \cdot m^2)/kg^2$$

$$F_a = Peso = m \cdot g$$



Leyes de fricción empíricas:

- La dirección de la fuerza de **fricción estática** entre cualesquiera dos superficies en contacto se oponen a la dirección de cualquier fuerza aplicada y puede tener valores:

- La dirección de la fuerza de **fricción cinética** que actúa sobre un objeto es opuesta a la dirección de su movimiento y está dada por:

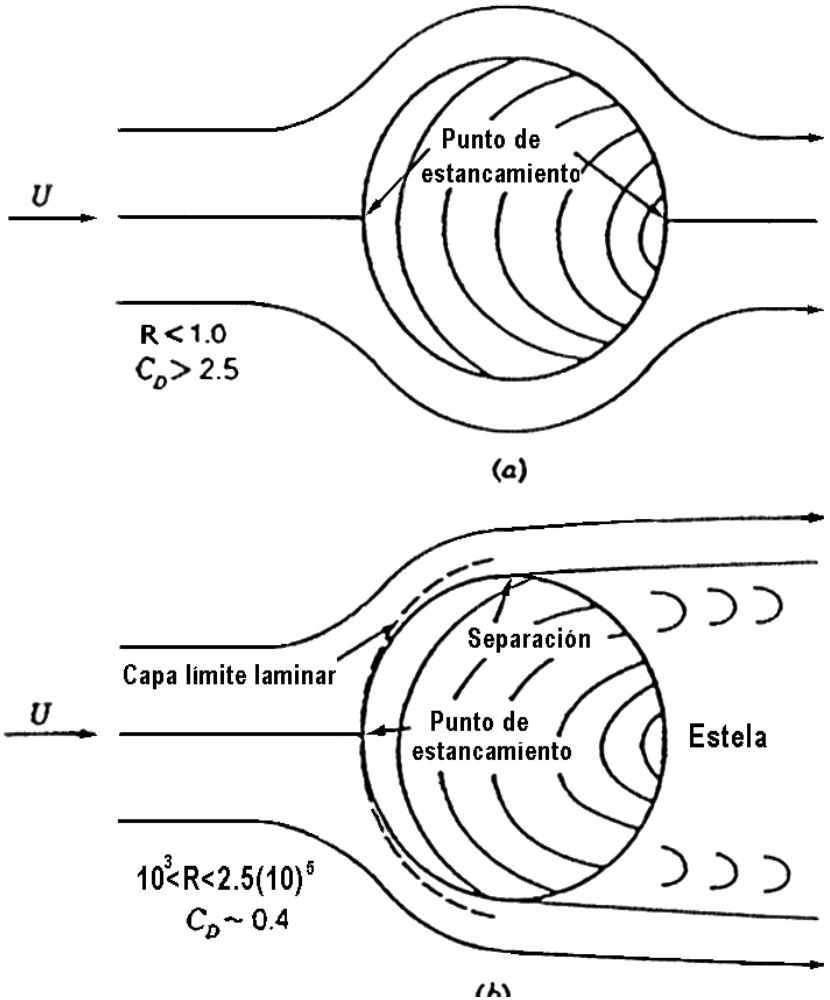
$$F_c = \mu_c \cdot n$$

μ_c es el coeficiente de fricción cinética.

Los coeficientes de fricción son prácticamente independientes del área de contacto.

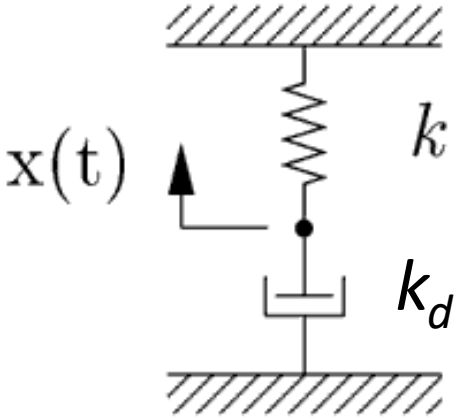
Dependen de los materiales	μ_e	μ_c
Acero sobre acero	0.74	0.57
Aluminio sobre acero	0.61	0.47
Cobre sobre cobre	0.53	0.36
Tela sobre cemento	1.0	0.8
Madera sobre madera	0.25 - 0.5	0.2
Vidrio sobre vidrio	0.94	0.4
Madera encerada sobre nieve húmeda	0.14	0.1
Madera encerada sobre nieve seca	---	0.04
Metal sobre metal (lubricados)	0.15	0.06
Hielo sobre hielo	0.1	0.03
Teflón sobre teflón	0.04	0.04
Articulaciones sinoviales en humanos	0.01	0.003

- Las fuerzas de fricción en fluidos pueden ser idealizadas utilizando un coeficiente de arrastre, C_D , que tendría en cuenta el tipo de superficie, la densidad y la viscosidad del fluido.



- La ley de Hook idealiza el comportamiento de los muelles, los modela utilizando una ley lineal.

$$F_s = -k_s \cdot x$$



- Muchos sistemas físicos se modelan utilizando una fuerza de restauración (muelle/spring) y una de fricción proporcional a la velocidad (amortiguador-damper)

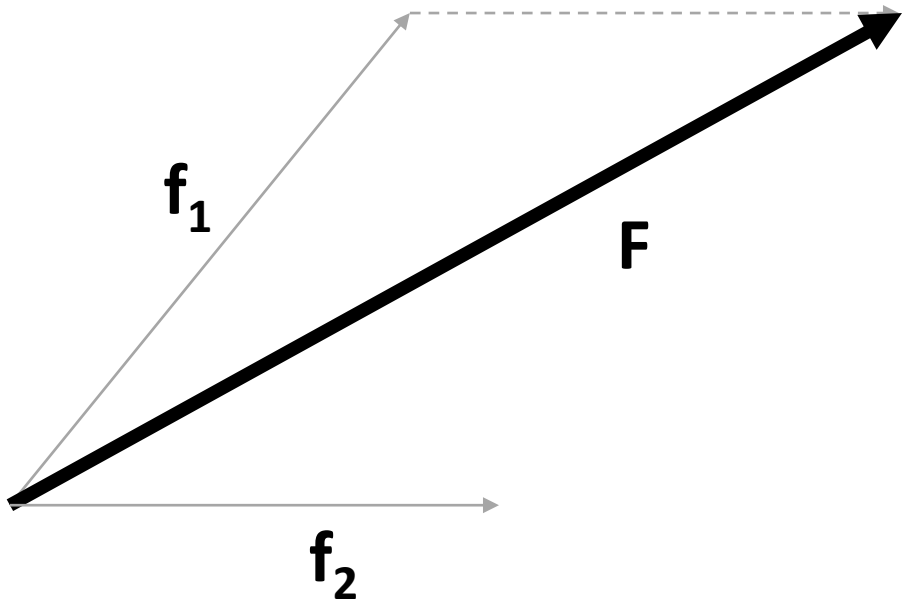
$$F_d = -k_d \cdot \dot{x}$$

- Añadiendo la inercia de la masa podremos representar el sistema mediante una ODE

$$m \cdot \ddot{x} = -k_s \cdot x - k_d \cdot \dot{x} + F_{ext} \Rightarrow$$

$$\boxed{m \cdot \ddot{x} + k_d \cdot \dot{x} + k_s \cdot x = F_{ext}}$$

- La fuerza resultante es la suma de todas las fuerzas
  - Cada fuerza es un vector
  - Magnitud y dirección
  - La fuerza resultante es también un vector



```
class Particle
{
    // Previous code

    // Accumulated force
    Vector3 force;

    // Clears accumulated force
    void clearForce();

    // Add force to apply in next integration only
    void addForce(const Vector3& f);
};
```

```
void Particle::clearForce()
{
    force.clear();
}
```

```
void Particle::addForce(const Vector3& f)
{
    force += f;
}
```

```
void Particle::integrate(float t)
{
    // Trivial case, infinite mass --> do nothing
    if (inverse_mass <= 0.0f) return;

    // Update position
    p += v * t;

    Vector3 totalAcceleration = a;
    a += force * inverse_mass;

    // Update linear velocity
    v += totalAcceleration * t;

    // Impose drag (damping)
    v *= powf(damping, t);

    clearForce();
}
```

Generadores

Fields

- Las fuerzas aplicadas solo tienen efecto un frame
  - Se borra el acumulador al final de la integración
- Hay que volver a aplicar la fuerza en cada frame
- Para fuerzas continuas resulta muy poco práctico

- Muchas fuerzas
- Con orígenes distintos
- Gravedad
- Por el comportamiento objeto
  - Rozamiento
- Por el entorno
  - Viento
  - Explosión
- Relación con otros objetos
- Otros tipos
  - Fuerza de aceleración de un tanque
  - Turbinas de un avión

- Comportamiento de fuerzas distinto
- Fuerza de gravedad fácil
  - Es constante
- La mayor parte están continuamente cambiando
  - Rozamiento cambia según el movimiento
  - Resistencia del aire aumenta cuando más rápido te mueves
  - La onda expansiva de la explosión va decayendo

- Solución genérica:

- Abstraer todo tipo de generadores de fuerzas
  - Tratarlos todos por igual
  - Simplifica el código para gestionarlos
  - Favorece crear nuevas fuerzas cambiando el menor código posible

- Registro de fuerzas
  - Se registran fuerzas en él
  - Se encarga de reaplicar el generador a las partículas cada frame

Generadores de fuerzas

- Interfaz (genérico)
- Cada generador será una clase derivada del interfaz. Uno (particular) por cada tipo de fuerza que exista.
  - Gravedad
  - Drag
  - Viento
  - …
- El generador aplicará la fuerza correspondiente en cada frame
- No requiere conocimiento específico por parte del cliente


```
class ParticleForceGenerator
{
    public:
        // Overload to provide functionality
        virtual void updateForce(Particle* particle, float t) = 0;
};
```

Código C++

```
class ParticleForceRegistry
{
protected:
    // Storage for generator-particle entry
    struct ParticleForceRegistration
    {
        Particle* particle;
        ParticleForceGenerator* fg;
    };

    typedef std::vector<ParticleForceRegistration> Registry;
    Registry registrations;

public:
    // Associate generator with a particle
    void add(Particle* particle, ParticleForceGenerator* fg);

    // Remove association
    void remove(Particle* particle, ParticleForceGenerator* fg);

    // Removes all associations. Particle and Generators won't be deleted
    void clear();

    // Update all the generators in the registry
    void updateForces(float t);
};
```


```
void ParticleForceRegistry::updateForces(float t)
{
    for (auto it = registrations.begin(), it != registrations.end(); ++it)
    {
        it->fg->updateForce(it->particle, t);
    }
}
```

Ejemplos simples

- Generador de gravedad
  - En lugar de incrustarlo en el update de la partícula
- Generador de drag
  - Resistencia aerodinámica
  - Más resistencia cuanta más velocidad lleve el objeto


```
class ParticleGravity: public ParticleForceGenerator
{
    // Acceleration for gravity
    Vector3 g;

public:
    ParticleGravity(const Vector3& gravity) : g(gravity) {}

    virtual void updateForce(Particle* particle, float t);
};


void ParticleGravity::updateForce(Particle* particle, float t)
{
    if (!particle->hasFiniteMass()) return;

    particle->addForce(g * particle->getMass());
}
```

$$f_{drag} = -\boldsymbol{v}\,(k_1|\hat{\boldsymbol{v}}| + k_2|\hat{\boldsymbol{v}}|^2)$$

```
class ParticleDrag: public ParticleForceGenerator
{
    // Coefficient for velocity
    float k1;

    // Coefficient for squared velocity
    float k2;

public:
    ParticleDrag(float _k1, float _k2): k1(_k1), k2(_k2) {}

    virtual void updateForce(Particle* particle, float t);
};
```


```
void ParticleDrag::updateForce(Particle* particle, float t)
{
    Vector3 f;
    particle->getVelocity(&f);

    // Drag coeffient
    float dragCoeff = f.normalize();
    dragCoeff = k1 * dragCoeff + k2 * dragCoeff * dragCoeff;

    // Final force
    f *= -dragCoeff;
    particle->addForce(f);
}
```