Game Physics

470420-1
Fall 2013
12/02/2013
Kyoung Shin Park
Multimedia Engineering
Dankook University

Application in Video Games

- 레이싱 게임: 자동차, 스노우 보드, etc..
  - Simulates how cars drive, collide, rebound, flip, etc..
- 스포츠 게임
  - Simulates trajectory of soccer, basket balls.
- First Person Shooters 게임: UnReal
  - Used to simulate bridges falling and breaking apart when blown up.
  - Dead bodies as they are dragged by a limb.
- 그 밖에:
  - Flowing flags / cloth.
- 실시간 물리는 매우 계산량이 많음. 그러나 최근 빠른 CPU를 사용하여 좀 더 구현이 쉬워지고 있음.

Definitions

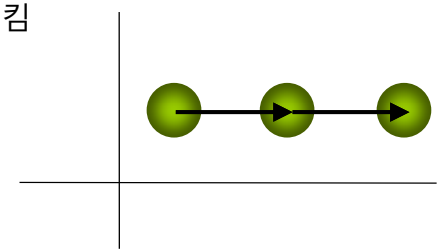
- 운동학 (Kinematics)
  - 힘이 배제된 운동의 변위, 속도와 가속도 등 **자세분석**
  - Study of movement over time.
  - Not concerned with the cause of the movement.
- 동역학 (Dynamics)
  - 물체의 운동 상태에 관한 **힘의 분석**
  - Study of forces and masses that cause the kinematic quantities to change as time progresses.

Game Physics

- 운동 (Motion)
- 위치 (Position), 속도 (Velocity), 가속도 (Acceleration)
- 힘 (Force), 중력 (Gravity)
- 부력(Buoyancy), 저항 (Drag)
- 마찰 (Friction)
  - 운동마찰 (Kinetic friction)
  - 정지마찰 (Static friction)
- 스프링 (Spring)

Motion

- 운동 (motion)은 시간에 따라 물체의 위치가 변하는 현상
- 운동을 나타내기 위해서 속력(speed), 속도 (velocity), 가속도 (acceleration)라는 물리량을 사용
- 속도 (velocity)는 벡터임
  - 벡터의 방향 (direction)은 움직임의 방향(direction)을 나타냄
  - 벡터의 길이 (magnitude)는 움직임의 속력(speed)를 나타냄
- 속도 벡터는 객체가 시간이 경과에 따라 얼마나 움직였나를 가리킴



Basic Motion

- 변위 (displacement) = 속도 (velocity) * 시간 (time)
- 만약 물체가 시작점 P_0에서 출발하여 일정한 속도 (constant velocity) v만큼 움직인다면, t 단위 시간이 경과한 후의 P(t) 위치(position):

$$P(t) = P_0 + v\ t$$

- NOTE: 단위(unit)는 거리의 경우 meters, 시간의 경우 seconds, 속도의 경우 meters/second를 의미함

Varying Velocity

- 앞의 공식은 물체가 일정한 속도로 움직일 때만 적용가능
  - 보다 일반적인 경우, 물체의 속도량은 시간이 경과에 따라 바뀜
- 속도는 단위 시간당 움직임 위치의 변화량
  - 시간에 따른 위치의 변화율
  - 속도는 위치를 시간에 대해 한번 미분한 값: $v(t) = \frac{d}{dt}P(t)$
  - 속도가 일정한 경우: $v(t) = v_0$
  - 속도가 일정한 가속으로 변하는 경우: $v(t) = v_0 + a\,t$
- 이때, 변위(displacement)는 속도의 적분 값 (integral)

$$displacement = \int_0^t velocity\ dt$$

Acceleration

- 가속도는 단위 시간당 속도의 변화량
- 가속도는 속도를 시간에 대해 한번 미분 값(derivative)

$$a(t) = \frac{d}{dt}v(t) = \frac{d^2}{dt^2}P(t)$$

- 속도는 가속도의 적분 값(integral)

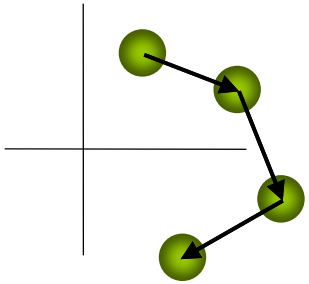
$$velocity = \int_0^t acceleration\ dt$$

Euler Method (or Euler Integration)

- **오일러 방법 (Euler method, 혹은 Euler Integration)**
  - 테일러 급수에서 유도된 방법으로 가장 기본적인 적분 방법
  - 그러나, 비교적 오차가 크게 남
  - 구간 [a, b]를 N개의 구간으로 나누었을 때 각각의 점을
    $t_i = a + i*h,\ i=0, 1, 2, .., N \quad \text{and} \quad h = (b-a)/N$
  - 오일러 방법은 $\omega_0=\alpha,\ \omega_{i+1} = \omega_i + h*f(t_i, \omega_i),\ i=0, 1, 2, .., N-1$
- 단위 시간별 물체가 직선으로 현재 속도 (current velocity)로 움직인 위치

$$dt = t_1 - t_0$$

$$P(t_1) = P(t_0) + v\ dt$$



Euler Method (or Euler Integration)

- 오일러 방법을 적용하여 현재 위치를 계산:

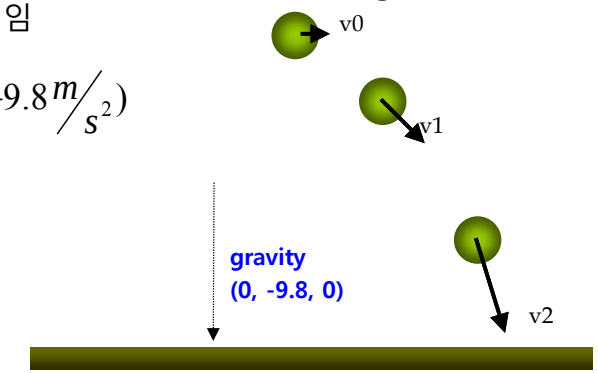
```
dt = t1 – t0;
Acc = ComputeAcceleration();
Vel = Vel + Acc * dt;
Pos = Pos + Vel * dt;
```

$$P(t) = \int_0^t (v_0 + at)dt = P_0 + v_0 t + \frac{1}{2} a\ t^2$$

Gravity

- 지구 중력 가속도 (gravity)는 지구의 표면에서 일정한 가속도인 9.8 meter/sec^2
- 중력장 내의 물체는 지구 중심 쪽으로 F=mg 힘을 받음. M은 질량(mass)임

$$F = mg\ \ (g = -9.8\ m/s^2)$$



Force

- 힘 (force)는 운동(motion)에 변화를 일으키는 원인
- **뉴튼 역학의 제 2법칙 (가속도의 법칙)**

$$F = ma$$

$$\Rightarrow a = F / m$$

- 만약 질량(mass) M인 물체에 힘 (force) F가 가해졌을 때, 오일러 방법을 적용하여 운동 (motion) 계산:

```
Acc = F/M;
Vel += Acc * dt;
Pos += Vel * dt;
Note that F, Acc, Vel, and Pos are all vectors. M is a scalar.
```

뉴튼 역학의 3법칙

1. 관성의 법칙: 모든 물체는 다른 물체의 움직임의 영향을 받지 않는다고 할 때, 정지해 있었다면 계속 정지해 있을 것이고, 움직이고 있었다면 일정한 속도로 계속 운동할 것이다.
2. 가속도의 법칙: 물체의 운동량의 변화율은, 크기와 방향에서, 그 물체에 작용하는 힘에 따른다.
3. 작용, 반작용의 법칙: 모든 작용에는 그 반대방향으로 같은 크기의 반작용이 존재한다.

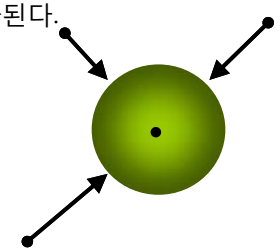
Gravitational Force

- 만유인력/중력 (gravitational force)
  - 이 우주 안에 존재하는 모든 물체들은 다른 물체를 무조건 끌어당기는 성질이 있는데, 그 힘의 크기는 두 물체의 질량의 곱에 비례하고, 둘 사이의 거리의 제곱에 반비례한다.
  - F의 방향은 큰 질량 물체의 무게중심 (center of mass)를 향한다.
- 물리 시뮬레이션
  - 보다 사실적인 시뮬레이션을 위해서 매 프레임마다 모든 물체간의 힘을 계산할 필요가 있다.
  - 여러 개의 힘이 적용될 때, 벡터가 추가된다.

$$F_{gravity} = \frac{GM_1M_2}{d^2} \quad (G = 6.673 \times 10^{-11})$$

$M_1, M_2 : mass(kg)$

$d : distance\ (meter)$



Projectile Motion

- 시간 t=0에서의 초기 위치가 P_0이고, 초기 속도가 v_0인 발사체 (projectile)의 위치

$$P(t) = P_0 + v_0 t + \frac{1}{2}gt^2$$

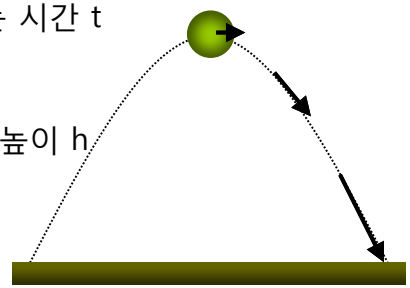
$$x(t) = x_0 + v_x t,\ \ y(t) = y_0 + v_y t - \frac{1}{2}gt^2,\ \ z(t) = z0 + v_z t$$

- 발사체가 최대 높이에 도달하는 시간 t

$$y(t) = v_y t - gt^2 = 0 \Rightarrow t = \frac{v_y}{g}$$

- 발사체가 도달할 수 있는 최대 높이 h

$$h = y_0 + \frac{v_y^{\ 2}}{2g}$$



Projectile Motion

- 발사체가 원래의 높이로 내려올 때까지 날아간 수평 거리

$$y(t) = y_0 + v_y t - \frac{1}{2}gt^2 = y_0 \Rightarrow t = 0 \text{또는} t = \frac{2v_y}{g}$$

$$x(t) = x_0 + v_x t \text{에} t \text{를 대입} \Rightarrow r = \frac{2v_x v_y}{g}$$

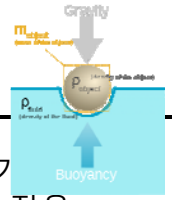
- 발사될 때의 초기 속력 s가 주어졌을 때, 발사체를 최대한 높이 올릴 수 있는 발사각도, θ

$$h = y_0 + \frac{v_z^{\ 2}}{2g} \Rightarrow h = y_0 + \frac{(s\sin\alpha)^2}{2g} \Rightarrow \alpha = \sin^{-1}\left(\frac{1}{s}\sqrt{2g(h-y_0)}\right)$$

- 원하는 도달 거리 r를 가기 위한 발사 각도, θ

$$r = \frac{2v_x v_y}{g} \Rightarrow r = \frac{2(s\cos\alpha)(s\sin\alpha)}{g} = \frac{s^2}{g}\sin 2\alpha \Rightarrow \alpha = \frac{1}{2}\sin^{-1}\frac{rg}{s^2}$$

Buoyancy



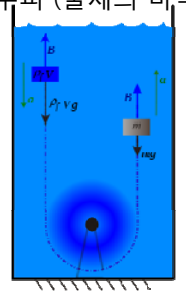
- 공기나 물과 같은 유체 속에 돌과 같은 고체가 경우, 중력 외에 부력과 저항력 두 가지 힘이 작용.
- 부력 (buoyancy force)
  - 밀도의 차이에 의하여 위로 저절로 상승하여 올라가는 부양력
  - **아르키메데스(Archimedes)의 원리: 물체에 작용하는 부력의 크기는 물체가 밀어낸 유체의 무게와 같다.**
  - 부력 = 유체의 밀도 x 중력가속도 (9.81 m/s^2) x 부피 (물체의 바닥 넓이 x 물체의 높이)

$$F_{buoyancy} = -\rho_f V g$$

ρ_f is the density of the fluid

V is the volume of the displaced body of liquid

g is the gravitatio nal acceleratio n



Drag

- 저항력 (drag force)
  - 물체가 움직임으로서 그를 방해하는 유체의 힘. 저항력은 물체가 물이나 공기 속에서 움직일 때에만 발생.
  - **Drag at low velocity (Stoke's drag): 저항력은 물체가 클수록 (r), 점성이 클수록 (h), 속도가 빠를 수록 (v)세다.**

    $F_d = -bv$

    $b = 6\pi\eta r$ (r : small spherical object radius, η : viscosity)
  - **Drag at high velocity:**

    $F_d = \frac{1}{2}\rho v^2 A C_d (v \bullet v)\frac{v}{\|v\|}$

    F_d is the force vector of drag

    ρ is the density of the fluid

    v is the velocity of the object relative to the fluid

    A is the reference area

    C_d is the drag coefficien t

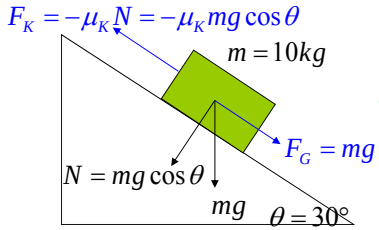
Kinetic Friction

- 운동마찰 (kinetic friction)
  - 서로 상대적으로 움직이는 두 표면 사이에서 발생하는 힘. 각자의 운동에 대한 저항으로 작용함.

    $F_K = -\mu_K N$

    N is the normal force

    μ_K is the coefficien t of kinetic friction



$F = F_G$(평행면으로의중력)$+ F_K$(운동마찰)

$a = F/m = (F_G + F_K)/m = g\sin\theta - \mu_K g\cos\theta$

Static Friction

- 정지마찰 (static friction)
  - 한 표면이 그 위에 놓여 있는 정지된 물체를 움직이지 못하도록 붙잡고 있는 힘. 물체에 가해지는 tangential force의 반대방향으로 작용하여 물체의 운동을 방해함.
  - 정지마찰력

    $F_S = -\mu_S N$

    N is the normal force

    μ_S is the coefficien t of static friction
  - 물체에 가해지는 힘이 F_S의 최대값을 넘는 순간 물체가 움직이기 시작하며, 그때부터는 F_S가 사라지고, F_K가 작용.
  - 정지된 물체를 움직이게 하는 것이 움직이는 물체를 계속 움직이게 하는 것보다 더 힘들다: $F_K < F_S$
  - 평면을 기울일 때 물체가 미끄러지기 시작하는 각도: 정지마찰력 = 평면과 수평인 중력성분

    $\mu_S mg\cos\theta = mg\sin\theta \Rightarrow \theta = \tan^{-1}\mu_S$

Momentum

- 운동량(momentum)은 질량(mass)과 속도(velocity)의 곱
- 입자의 운동량(momentum) 변화의 속도는 입자에 작용하는 힘(force)에 비례하고, 힘의 방향과 일치한다.
- 힘(force)은 운동량(momentum), P,의 미분 값

  $P = mv$

  $\Rightarrow \frac{dP}{dt} = m\frac{dv}{dt} = ma = F$

- Momentum을 사용한 운동 변위 계산

```
Force = ComputeTotalForce();
Momentum += Force * dt;          // 힘을 적분하여 운동량 계산
Velocity = Momentum / Mass;      // 운동량과 질량으로 속도 계산
Position += Velocity* dt;        // 속도를 적분하여 위치 계산
```

Angular Velocity

- 각속도 (angular velocity)는 물체의 회전속도
- 각속도는 시간당 각도의 변화율 (radian/seconds 단위)

$$\omega(t) = \frac{d}{dt}\theta(t)$$

- 각속도는 회전축 A에 평행이고 크기가 w(t)인 벡터

$$\omega(t) = \omega(t)A$$

- 회전의 중심으로부터 r만큼 떨어진 곳에서 물체가 속도 v로 운동하고 있을 때
  - 물체의 속도: $v(t) = |\omega(t)r|$
  - 물체의 위치를 r(t)라고 하면 물체의 선속도 (linear velocity):

$$v(t) = \omega(t) \times r(t)$$

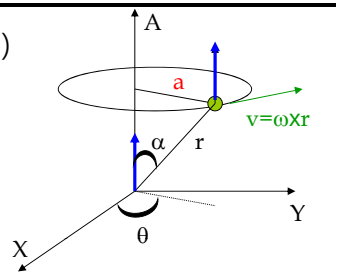
Centrifugal Force

- 물체의 선가속도 (linear acceleration)

$$a(t) = \omega'(t) \times r(t) + \omega(t) \times r'(t)$$
$$= \omega'(t) \times r(t) + \omega(t) \times [\omega(t) \times r(t)]$$

- 각속도가 일정한 경우: w'(t)=0

$$a(t) = \omega(t) \times [\omega(t) \times r(t)]$$



  - 가속도 a는 안쪽 방향임: 끝의 장력 (tension)으로부터 발생
  - 물체에서 장력과 같은 크기의 반대 방향으로 힘이 작용 - 원심력 (centrifugal force):

$$F_c = -m(\omega(t) \times [\omega(t) \times r(t)])$$

  - r(t)와 w(t)가 수직인 경우, 원심력은 한 scalar로 표현됨

$$F_c = m\omega^2 r = \frac{mv^2}{r}$$

Rigid Motion

- 강체 운동 (rigid motion)
  - 강체(rigid body)란 몸의 **크기 변형이 없는** 단단한 물체(solid object)이다. (오로지 translation & rotation만 가능)
- 강체 동역학 (Rigid body dynamics)
  - Linear & angular position, velocity, acceleration

```
Force = ComputeTotalForce();
Momentum += Force * dt;
Velocity = Momentum / Mass;
Position += Velocity* dt;
Torque = ComputeTotalTorque();
AngMomentum += Torque * dt;
Matrix I = Matrix*RotInertia*Matrix.Inverse(); // tensor
AngVelocity = I.Inverse()*AngMomentum;
Matrix.Rotate(AngVelocity*dt);
```

Integration Method

- 오일러 방법 (Euler method)
  - $v = v_0 + a*dt$, $x = x_0 + v*dt$
  - 변화율이 상수일 때는 100%정확함
  - 변화율이 시간에 따라 변할 때는 에러가 존재함

```
float t = 0;   // 현재 시간
float dt = 1; // 시간 간격 (timestamp)
float velocity = 0;      // 초기 속도
float position = 0;      // 초기 위치
float force = 10;
float mass = 1;
float acceleration = force/mass;
while (t<=10) {
   position += velocity * dt;
   velocity += acceleration * dt;
   t += dt;
}
```

Initial : $y'(t) = f(t, y(t)), y(t_0) = y_0$

Euler Method : $y_{n+1} = y_n + hf(t_n, y_n)$

Integration Method

- 룽게-쿠타 방법 (Runge-Kutta method)
  - RK4는 매우 정확하고 미분에 안정적임

Initial : $y' = f(t, y), y(t_0) = y_0$

RK4: $y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

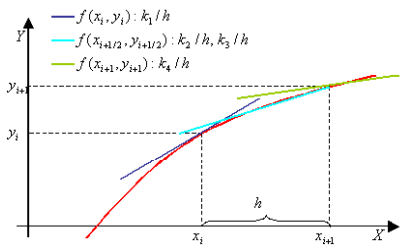
$k_1 = f(t_n, y_n)$

$k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$

$k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2)$

$k_4 = f(t_n + h, y_n + hk_3)$

$slope = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$



Integration Method

```
void RK4Integration(vector3& pos, vector3& vel, float t, float dt) {
    vector3 k1Vel = vel;
    vector3 k1Acc = f(t, pos, vel);
    vector3 k2Vel = vel + 0.5f * dt * k1Acc;
    vector3 k2Acc = f(t + 0.5f * dt, pos + 0.5f * dt * k1Vel, k2Vel);
    vector3 k3Vel = vel + 0.5f * dt * k2Acc;
    vector3 k3Acc = f(t + 0.5f * dt, pos + 0.5f * dt * k2Vel, k3Vel);
    vector3 k4Vel = vel + dt * k3Acc;
    vector3 k4Acc = f(t + dt, pos + dt * k3Vel, k4Vel);
    pos += (dt / 6.0f) * (k1Vel + 2.0f * k2Vel + 2.0f * k3Vel + k4Vel);
    vel += (dt / 6.0f) * (k1Acc + 2.0f * k2Acc + 2.0f * k3Acc + k4Acc);
}
while (t<=10) {
    RK4Integration(position, velocity, t, dt);
    t += dt;
}
```

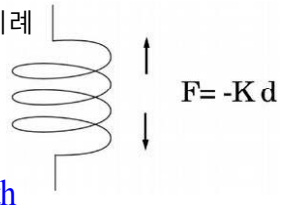
Springs

- Hooke's Law
  - 스프링의 힘은 스프링의 길이/변위에 비례

$F = -K_s d$

K_s is the spring constant

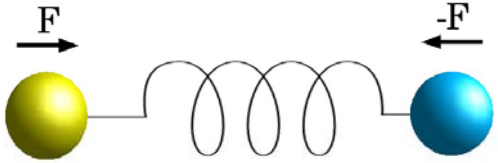
d is the displaceme nt from rest length



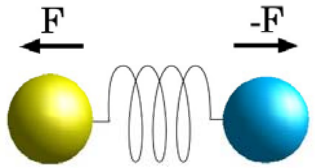
- 스프링은 스프링으로 연결된 2개의 질점(2 point mass)으로 모델링함.
- 동일한 크기의 반대방향의 힘이 양쪽에 적용됨.

Springs

- 스프링의 입자가 멀리 떨어질 수록(stretched) 스프링의 안정상태 위치로 끌어 당기는 힘이 커짐



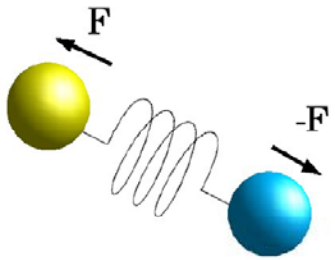
- 스프링의 양끝을 동시에 눌렀을 때(compressed) 입자의 위치가 스프링의 안정상태에서의 길이만큼 떨어지도록 미는 힘이 작용함



Springs

- 두 점 간의 벡터를 사용하여 변위와 힘의 방향을 계산

```
Vector3 v = point1 – point0;
float displacement = v.length() – restLength;
v.normalize();
Vector3 force = springConstant * displacement * v;
```



Spring Classes

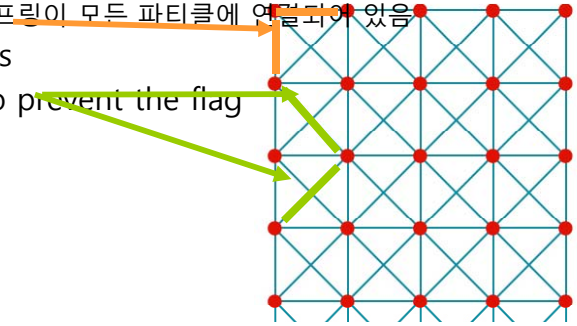
```
class PointMass
{
    float mass;
    float position[3];
    float velocity[3];
    float acceleration[3];
    void ClearForces();
    void AddForce();
    void Update();
    void Freeze();
}
```

Spring Classes

```
class Spring
{
    float pointMass[2];
    float springConstant;
    float restLength;
    void Update();
}
```

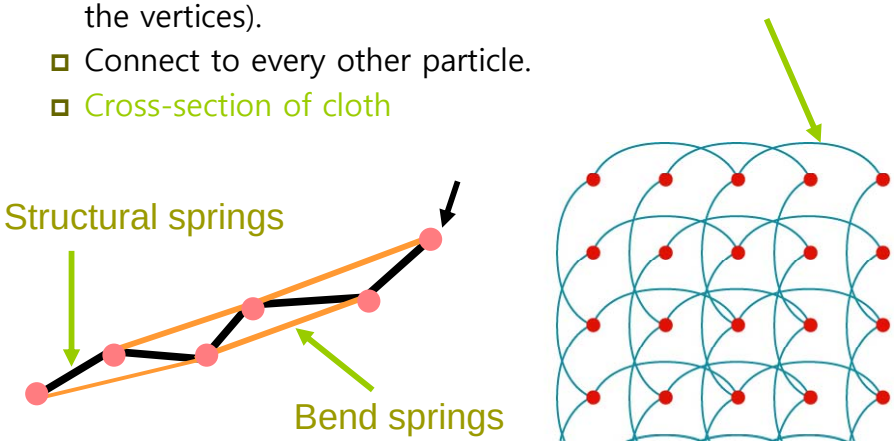
Simulating Cloth

- 옷감/천(Cloth)은 스프링 매쉬로 시뮬레이션 할 수 있음
  - 3가지 종류의 스프링이 모든 파티클에 연결되어 있음
- Structural Springs
- Shear Springs (to prevent the flag from shearing)



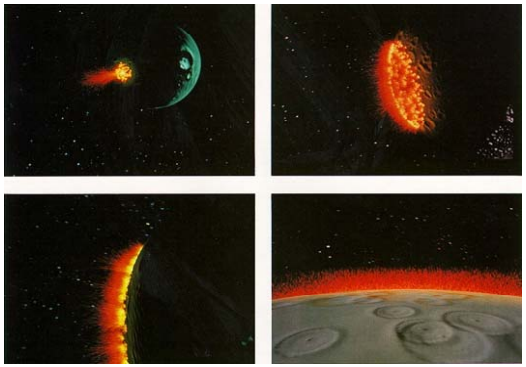
Simulating Cloth

- Bend Springs (to prevent the flag from folding along the vertices).
- Connect to every other particle.
- Cross-section of cloth



Particle Systems

- Star Trek II (1983) "Genesis Effect"에서 처음으로 사용됨



Particle Systems

- 입자시스템 (Particle systems)은 **폭발, 연기, 화재, 스프레이** 등을 시뮬레이션
- **젤리나 천** 등과 같은 비강체 객체 모델링하는데 유용
- **질량(Mass), 위치(Position), 속도(Velocity)**를 가진 무한히 작은 물체
- 뉴턴 입자의 움직은 다음과 같이 적용:
  - F=ma (F=force, m=mass, a=acceleration)
  - a=dv/dt (Change of velocity over time- v=velocity; t=time)
  - v=dp/dt (Change of distance over time- p=distance or position)
  - F, m, v, p로 구성된 기본 데이터 구조를 가짐

E.g. a 3D particle might be represented as:

```
class Particle
{
    float mass;
    float position[3];// [3] for x,y,z components
    float velocity[3];
    float forceAccumulator[3];
}
```

- forceAccumulator 가 있는 이유는 입자가 여러가지 힘에 의해 영향을 받을 수 있기 때문 - e.g. 축구공은 중력에 영향을 받고 누군가가 공을 하는 외부 힘에 영향을 받음.
- 무엇인가 입자에 전달된 힘이 있다면 단순히 이 힘 (X, Y, Z)를 forceAccumulator에 더 추가해주면 됨.

E.g. 3D Particle System

```
class ParticleSystem
{
    particle *listOfParticles;
    int numParticles;
    void EulerStep();// Discussed later
}
```

Particle Dynamics Algorithm

For each particle

{

- 입자에 작용하는 힘 계산 (Compute the forces that are acting on the particle)
- 입자의 가속도 계산 (Compute the acceleration of each particle) F=ma; a=F/m
- 입자의 속도 계산 (Compute velocity of each particle due to the acceleration)
- 입자의 위치 계산 (Compute the new position of the particle based on the velocity)

}

How do you calculate velocity?

- Recall that:
  - a = dv/dt (ie change in velocity over time)
  - v = dp/dt (ie change in position over time)
- 속도 (velocity)는 가속도의 적분 (integral of acceleration)
- 위치 (position)은 속도의 적분 (integral of velocity)
- 가장 단순한 수치 통합 방법 (**Euler's Method**):
  - **Q(t+dt) = Q(t) + dt * Q'(t)**
  - So in our case:
    - To find velocity at each simulation timestep:
      - **v(t+dt) = v(t) + dt * v'(t) = v(t) + dt * a(t) // we know a(t) from F=ma**
    - To find the position at each simulation timestep:
      - **p(t+dt) = p(t) + dt * p'(t) = p(t) + dt * v(t) // we know v(t)**

E.g. Euler Integration (EulerStep)

- To find velocity at each simulation timestep:

```
v(t+dt) = v(t) + dt * a(t) // we know a(t) from F=ma
v_next[x] = v_now[x] + dt * a[x];
v_next[y] = v_now[y] + dt * a[y];
v_next[z] = v_now[z] + dt * a[z];
```

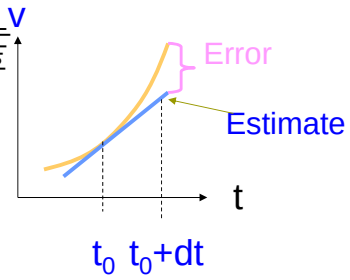
- To find the position at each simulation timestep:

```
p(t+dt) = p(t) + dt * v(t) // we know v(t)
p_next[x] = p_now[x] + dt * v_now[x];
p_next[y] = p_now[y] + dt * v_now[y];
p_next[z] = p_now[z] + dt * v_now[z];
```

- Remember to save away v_next for the next step through the simulation:
  - v_now[x] = v_next[x]; v_now[y] = v_next[y]; v_now[z] = v_next[z];

Warning about Euler Method

- 시간 단계가 크면 (big time steps) 적분 에러가 발생함
- 이 에러가 발생되면 입자가 통제되지 않고 무한 값으로 달아가버리는 현상이 나타남
- 따라서 작은 시간 단계를 사용하라 – 하지만 작은 시간 단계를 사용하면 CPU 시간을 많이 잡아먹을 수 있음
- 매 시간 단계마다 DRAW를 할 필요는 없음 - E.g. 10 timesteps를 계산한 후 그 결과를 그리도록 함
- 더 나은 해결책:
  - Adaptive Euler Method
  - Midpoint Method
  - Implicit Euler Method
  - Runge Kutta Method



Adaptive Step Sizes

- 가능한 적은 계산을 할 수 있도록 하는 가장 이상적으로 큰 step-size (dt)를 원함
- 그러나, 더 큰 step-size는 더 많은 오류를 포함하고 결국 시스템을 불안정하게 할 수 있음
- 따라서, 일반적으로 작은 step-size를 필요로 함 – 불행하게도 좀 작은 step-size는 시간이 오래 걸리 수 있음
- 작은 step-size를 모든 시간에 가능하면 강제하지 않도록 함

Euler with Adaptive Step Sizes

- Suppose you compute 2 estimates for the velocity at time t+dt:
- So v1 is your velocity estimate for t+dt
- And v2 is your velocity estimate if you instead took 2 smaller steps of size dt/2 each.
- Both v1 and v2 differ from the true velocity by an order of dt^2 (because Euler's method is derived from Taylor's Theorem truncated after the 2nd term- see reference in the notes section of this slide)
- By that definition, v1 and v2 also differ from each other by an order of dt^2
- So we can write a measure of the current error as: E = |v1-v2|
- Let $E_{tolerated}$ be the error that YOU can tolerate in your game.
- Adaptive step size dt_{adapt} is calculated as approximately:

$$dt_{adapt} = Sqrt(E_{tolerated} / E) * dt$$

- So a bigger tolerated error would allow you to take a bigger step size. And a smaller one would force a smaller step size.

Handling Collisions

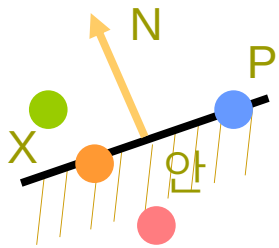
- 입자들 (particles)는 표면에 닿으면 튕겨나와야 함
  1. 충돌 (collision)이 발생했을 때 감지해야 함
  2. 충돌 (collision)에 대한 올바른 반응을 결정해야 함

Detecting Collision

- 일반적으로 충돌 문제는 복잡함
  - Particle/Plane Collision – 가장 간단함
  - Plane/Plane Collision
  - Edge/Plane Collision

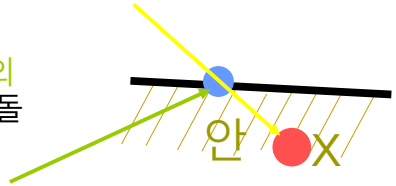
Particle/Plane Collisions

- P = 평면(plane) 위에 한 점
- N = 평면(plane)의 법선벡터 (normal)
- X = 입자의 지점
- For **(X – P) . N**
  - If > 0 then X는 평면의 밖에 존재함
  - If = 0 then X는 평면 위에 존재함
  - If < 0 then X는 평면의 안쪽에 존재함



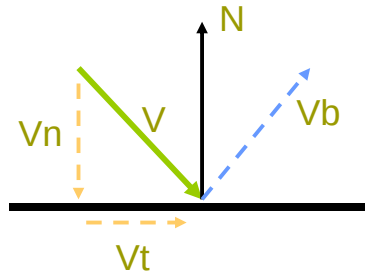
Collision Response – dealing with the case where particle penetrates a plane (and it shouldn't have)

- 만약 입자 X가 안쪽으로 들어온 경우, 평면(plane)의 표면으로 움직여 주고, 충돌 반응을 계산함



Collision Response

- N= (충돌하는) 평면의 법선 벡터
- Vn= V의 수직성분 (normal component of a vector V)
  Vn= (N . V) V
- Vt= V의 수평성분 (tangential component of a vector V)
  Vt=V-Vn
- Vb= (충돌에) 바운스한 반응 벡터
  Vb=(1 – Kf) * Vt – (Kr * Vn)
  - Kr=반발계수 (coefficient of restitution): 즉, 표면이 얼마나 탄력적인지. 1=perfectly elastic; 0=stick to wall.
  - Kf= 마찰계수 (coefficient of friction): 즉, 바운스 한 후에 Vt가 얼마나 느려지는지. 1=particle stops in its tracks. 0=no friction.



References

- http://www.evl.uic.edu/spiff/class/cs426/Notes/physics.ppt
- http://en.wikipedia.org/wiki/Newton%27s_laws_of_motion
- http://en.wikipedia.org/wiki/Equations_of_motion
- http://en.wikipedia.org/wiki/Projectile
- http://en.wikipedia.org/wiki/Trajectory
- http://en.wikipedia.org/wiki/Buoyancy
- http://en.wikipedia.org/wiki/Drag_(physics)
- http://en.wikipedia.org/wiki/Euler_method
- http://en.wikipedia.org/wiki/RK4
- http://www.gaffer.org/game-physics/