물리기반 모델링

비탄성 충돌

동명대학교
강영민

탄성 계수

- 충격량(impulse)
  - J
    - 운동량의 변화
    - $|J| = m(v_+ - v_-)$
- 탄성계수
  - $$\epsilon = \frac{-(v_{1+} - v_{2+})}{v_{1-} - v_{2-}}$$

충격량과 탄성의 관계

- 3개의 식이 필요

$$|\mathbf{J}| = m_1(v_{1+} - v_{1-})$$

$$-|\mathbf{J}| = m_1(v_{2+} - v_{2-})$$

$$\epsilon = \frac{-(v_{1+} - v_{2+})}{v_{1-} - v_{2-}}$$

$$v_{1+} = |\mathbf{J}|/m_1 + v_{1-}$$

$$v_{2+} = -|\mathbf{J}|/m_1 + v_{2-}$$

$$\epsilon = \frac{-(v_{1+} - v_{2+})}{v_{1-} - v_{2-}}$$

$$\epsilon(v_{1-} - v_{2-}) = -(v_{1+} - v_{2+})$$

충격량

- 충격량과 충돌이전 속도의 관계

$$\epsilon(v_{1-} - v_{2-}) = -(|\mathbf{J}|/m_1 + v_{1-} + |\mathbf{J}|/m_2 - v_{2-})$$
$$\epsilon(v_{1-} - v_{2-}) = -(|\mathbf{J}|(1/m_1 + 1/m_2) + v_{1-} - v_{2-})$$

- 충격량의 크기

$$|\mathbf{J}| = (1 + \epsilon)(v_{1-} - v_{2-})/(1/m_1 + 1/m_2)$$

속도의 갱신

- 충돌한 방향으로 속도의 갱신

$$v_{1+} = v_{1-} + |\mathbf{J}|/m_1$$

$$v_{2+} = v_{2-} - |\mathbf{J}|/m_2$$

충돌처리

```
void CDynamicSimulator::collisionHandler(int i, int j) {
    // collision detect
    CVec3d p1; p1 = particle[i].getPosition();
    CVec3d p2; p2 = particle[j].getPosition();
    CVec3d N ; N = p1 - p2;
    double dist = N.len();
    double e = 0.1;
    if(dist < particle[i].getRadius() + particle[j].getRadius()) {

        double penetration = particle[i].getRadius() + particle[j].getRadius() - dist;

     // collision detected
     N.normalize();
     CVec3d v1; v1 = particle[i].getVelocity();
     CVec3d v2; v2 = particle[j].getVelocity();
     double v1N = v1 ^ N; // velocity along the line of action
     double v2N = v2 ^ N; // velocity along the line of action
     double m1 = particle[i].getMass();
     double m2 = particle[j].getMass();
     // approaching ?
     if( v1N-v2N < 0 ) { // approaching
          double vr = v1N - v2N;
          double J = -vr*(e+1.0)/(1.0/m1 + 1.0/m2);
          double v1New = v1N + J/m1;
          double v2New = v2N - J/m2;
          v1 = v1 - v1N * N + v1New*N;
          v2 = v2 - v2N * N + v2New*N;
          particle[i].setVelocity(v1.x, v1.y, v1.z);
          particle[j].setVelocity(v2.x, v2.y, v2.z);
     }
     p1 = p1 + ((1.0+e)*penetration)*N;
     p2 = p2 - ((1.0+e)*penetration)*N;
     particle[i].setPosition(p1.x, p1.y, p1.z);
     particle[j].setPosition(p2.x, p2.y, p2.z);
    }
}
```

다수의 입자 만유인력

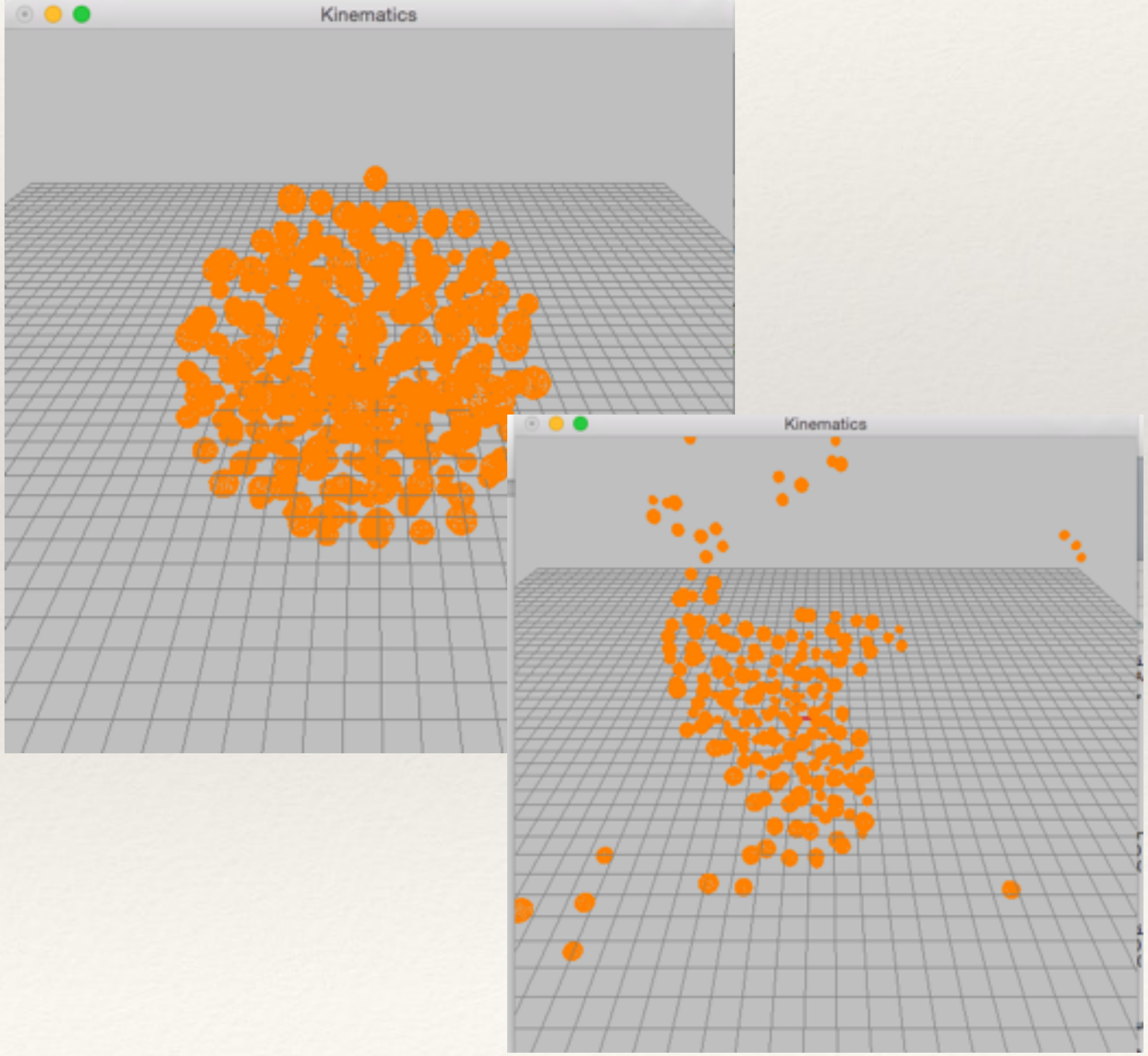
- 랜덤하게 입자를 생성
- 입자간 인력 작용
- 인력의 크기
    - $\dfrac{m_i m_j}{r^2}$ 에 비례

인력 계산

```
CVec3d CDynamicSimulator::computeAttraction(int i, int j) {
    // collision detect
    CVec3d xi; xi = particle[i].getPosition();
    CVec3d xj; xj = particle[j].getPosition();
    CVec3d xij; xij = xj-xi;
    double dist = xij.len();
    xij.normalize();
    double mi = particle[i].getMass();
    double mj = particle[j].getMass();

    double G = 5.5;
    CVec3d force;
    force = (G*mi*mj/(dist*dist))*xij;
    return force;

}
```

시뮬레이션

```
void CDynamicSimulator::doSimulation(double dt, double currentTime) {

    if(dt>0.01)dt=0.01; // maximum dt

    CVec3d forcei;
    CVec3d forcej;
    for (int i=0; i<NUMPARTS; i++) {
        for (int j=i+1; j<NUMPARTS; j++) {
            forcei = computeAttraction(i, j);
            forcej = -1.0*forcei;
            particle[i].addForce(forcei);
            particle[j].addForce(forcej);
        }
    }

    for (int i=0; i<NUMPARTS; i++) {
        particle[i].simulate(dt, currentTime);
    }

    for (int i=0; i<NUMPARTS; i++) {
        for (int j=i+1; j<NUMPARTS; j++) {
            collisionHandler(i, j);
        }
    }

}
```


Kinematics
Kinematics