Molekuláris dinamika

10. előadás

Miről is szól a MD?

- nagy részecskeszámú rendszerek
- ismerjük a törvényeket mikroszkópikus szinten

? **Hogyan tudjuk megérteni a folyadékok, gázok, szilárdtestek makroszkópikus viselkedését?**

minden részecske mozgását szimuláljuk → **MOLEKULÁRIS DINAMIKA**

a részecskék mozgásegyenleteit integráljuk numerikusan

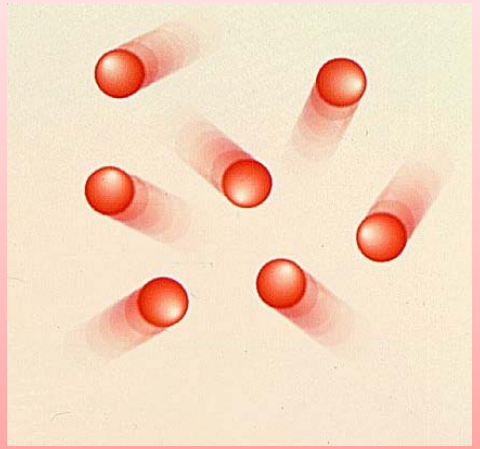
ha a rendszer egyensúlyban van, a hőmérséklet, nyomás és más makroszkopikus mennyiség meghatározható az időbeli átlagokból

ALAPOK: A. Rahman, Phys. Rev. **136**, A405 (1964); L. Verlet, Phys. Rev. **159**, 98 (1967)

!!!

RECEPT:

- végy N pontszerű részecskét;
- használj klasszikus mechanikát;
- a pontoknak adj tömeget, pozíciót és sebességet;
- a pontok kölcsönhatását modellezd potenciálokkal, ezekből számolj erőket;
- a rendszer időbeli fejlődését megkapod a Newton egyenletek integrálásával.



Történeti áttekintés

kontinuum

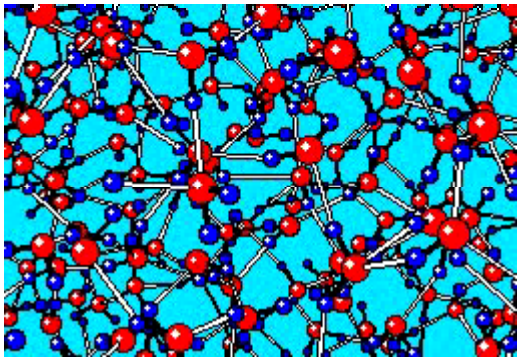
- 1500-1600: L. da Vinci, Galileo Galilei
  1700-1800: Euler, Bernoulli
  merev testek, rudak
  (parciális differenciálegyenletek, kontinuum elméletek)
- **Kontinuum mechanikai elmélet**
- 1930: **törések, diszlokációk elméletének kifejlesztése**
- 1960-1970: **végeselem módszer**
- 1990: **végeselem módszerek és molekuláris dinamikai módszerek egyesítése**

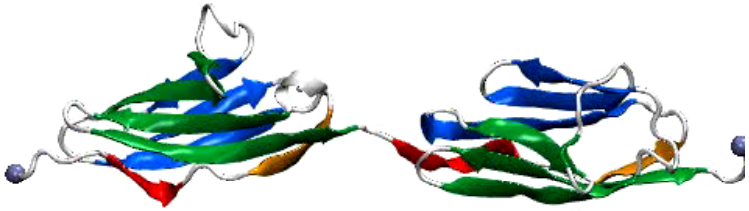
atomi szint

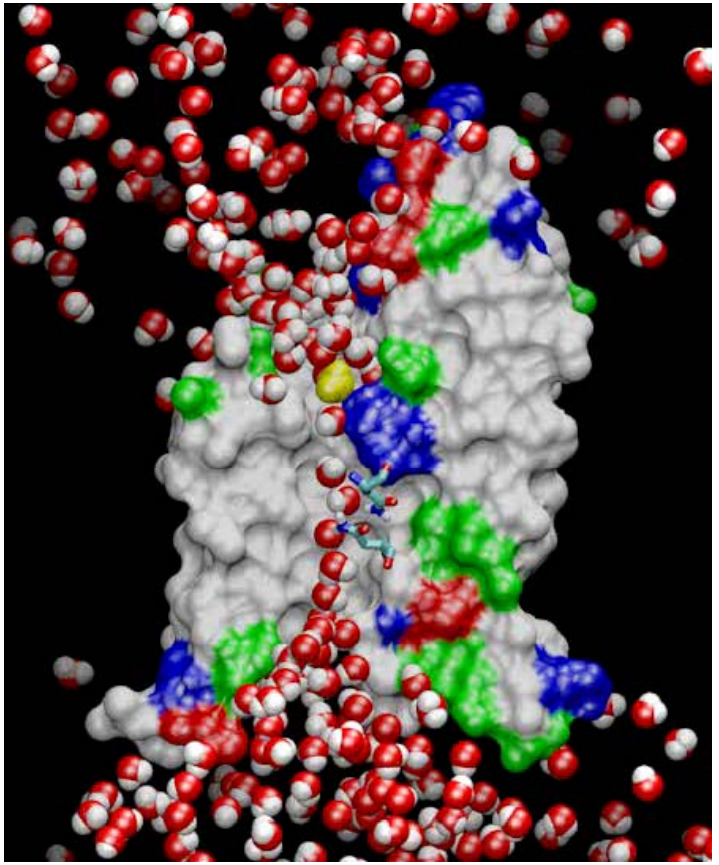
- XX. század: **atomok felfedezése**
- 1950-es évek vége:
  Alder és Wainwright,
  molekuláris dinamika,
  egyszerű folyadékok viselkedése
- 1964: Rahman,
  folyékony argon szimulációja
  valószerű potenciálokkal
- 1960-1980: Kohn-Sham, **numerikus módszerek**, pl. DFT
- 1974: Stillinger és Rahman,
  valós rendszer (**folyékony víz**)
  első MD szimulációja
- 1980: Yip
  1990: Abraham és társai
  első **repedés- és törésszimuláció**

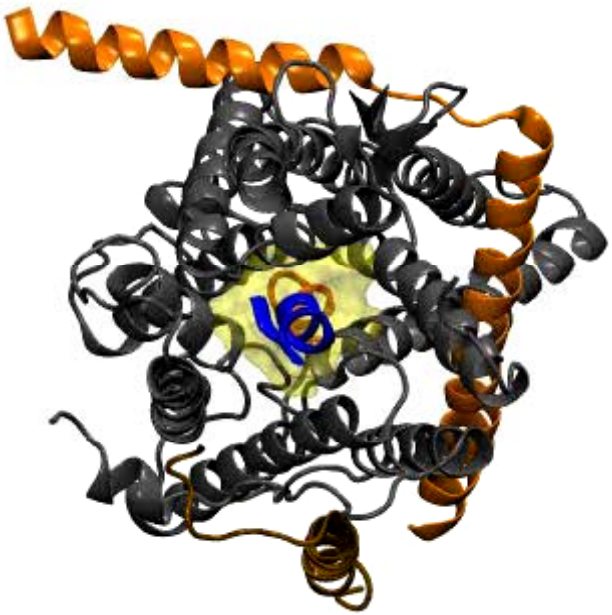
- biofizikai rendszerek, törések, deformációk MD szimulációja rutin feladat
- a szimulációs technikák száma robbanásszerűen megnőtt:
  sajátos problémák sajátos módszerekkel,
  vegyes kvantummechanikai/klasszikus mechanikai szimulációk
  enzimreakciók és törések szimulációjára

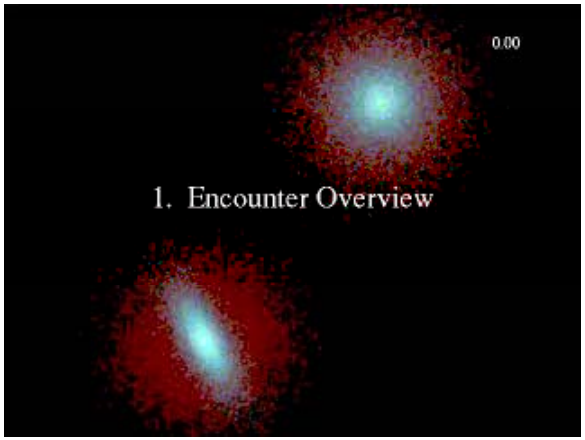
Példák MD szimulációkra

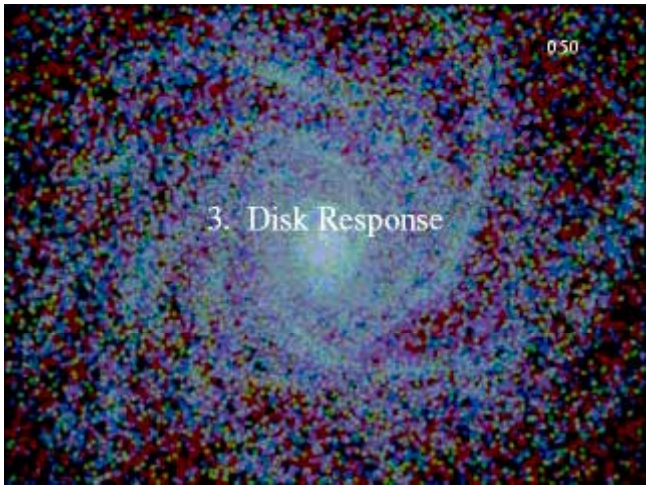


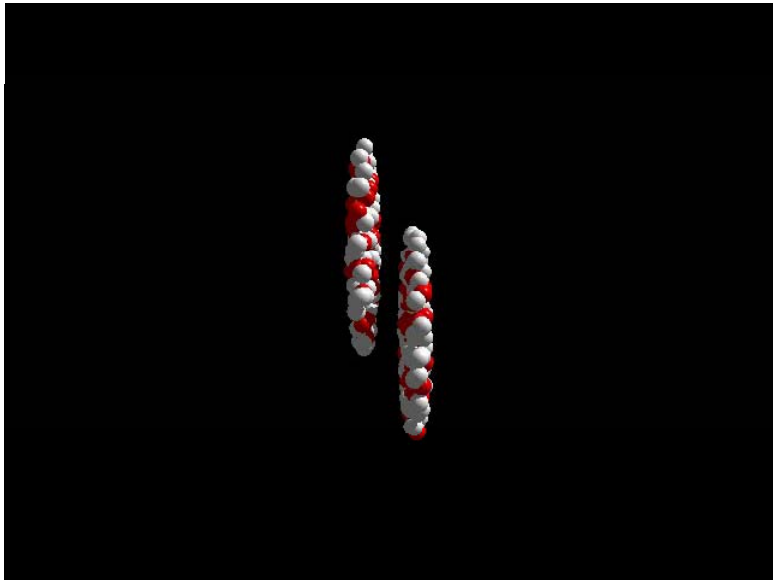










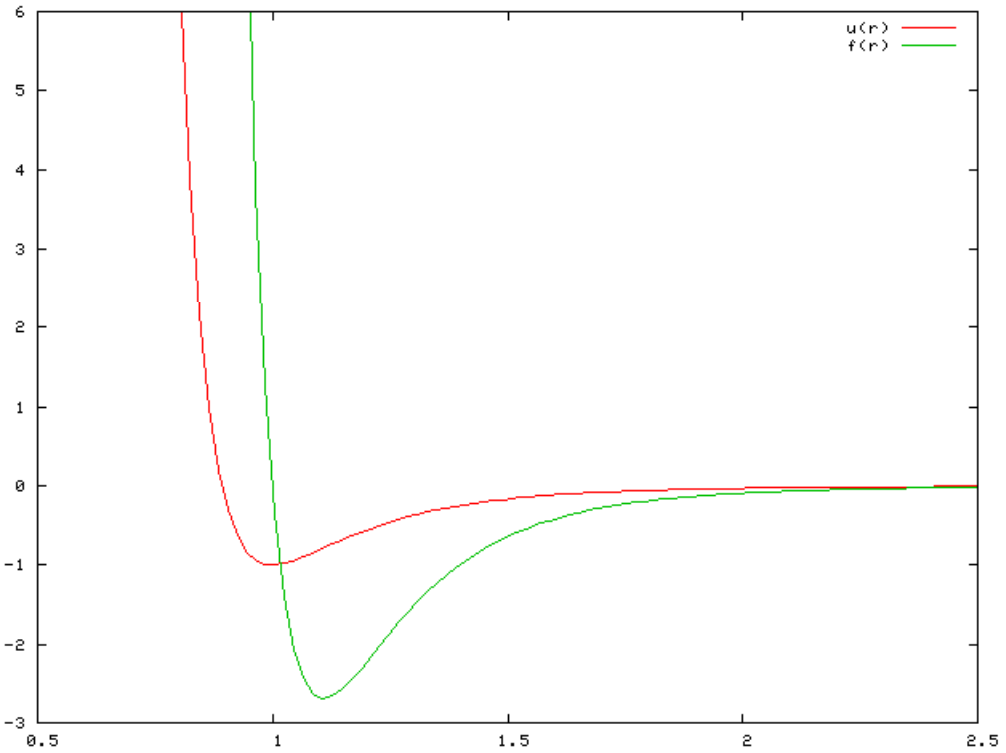


Az intermolekuláris potenciál

az argon szimulációja:

- klasszikus dinamika
- gömb alakú atomok
- kémiai kölcsönhatás nincs
- a belső struktúrától eltekintünk
- párkölcsönhatás:
  magok taszítása + van der Waals vonzás

$$U(r) = u(r_{12}) + u(r_{13}) + \ldots + u(r_{23}) + \ldots = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} u(r_{ij})$$



Lennard-Jones potenciál

$$u(r) = \epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - 2 \left(\frac{\sigma}{r} \right)^{6} \right]$$

Lennard-Jones erő

$$F(r) = \frac{-dU(r)}{dr} = 12 \frac{\epsilon}{\sigma} \left[\left(\frac{\sigma}{r} \right)^{13} - \left(\frac{\sigma}{r} \right)^{7} \right]$$

Mennyiségek

általában úgy választjuk, hogy ne kelljen túl nagy/kicsi számokkal dolgozni

egységnek választjuk: $\sigma = 1;\ \varepsilon = 1;\ m = 1.$

átalakítás:

mennyiség	egység	érték az Argon esetében
távolság	σ	$3.4\text{x}10^{-10}\,m$
energia	ϵ	$1.65\text{x}10^{-21}\,J$
tömeg	m	$6.69\text{x}10^{-26}\,kg$
idő	$\sigma\left(m/\epsilon\right)^{1/2}$	$2.17\text{x}10^{-12}\,s$
sebesség	$\left(\epsilon/m\right)^{1/2}$	$1.57\text{x}10^{2}\,m/s$
erő	ϵ/σ	$4.85\text{x}10^{-12}\,N$
nyomás	ϵ/σ^2	$1.43\text{x}10^{-2}\,Nm^{-1}$
hőmérséklet	ϵ/k_B	120K

pl. 2000 időlépés, $\Delta t = 0.01$ → $t = 20 = 4.34\text{x}10^{-11}$s

tipikus szimulálási idők: **10^{-11} – 10^{-9} s .... 10^{-6}s**

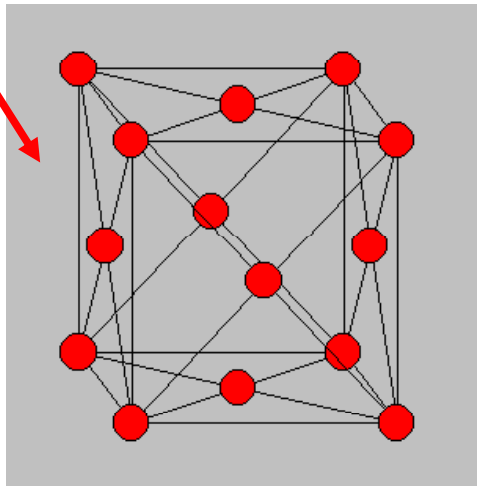
Inicializálás

használt változók:

```
const int N = 64;          // number of particles
double r[N][3];            // positions
double v[N][3];            // velocities
double a[N][3];            // accelerations
```

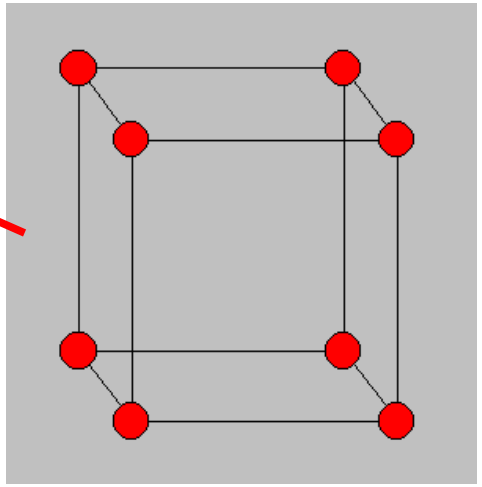
a kezdeti konfiguráció közel kell legyen az egyensúlyi konfigurációhoz!!!

sűrű rendszer: kezdeti helyzetek lapcentrált köbös rács rácspontjaiban
az adott hőmérsékletnek megefelelő véletlenszerű sebességek



első verzióban egyszerű köbös rácsra inicializáljuk az atomokat:

```
double L = 10;             // linear size of cubical volume
double vMax = 0.1;         // maximum initial velocity component
void initialize() {

    // initialize positions
    int n = int(ceil(pow(N, 1.0/3)));  // number of atoms in each direction
    double a = L / n;                  // lattice spacing
    int p = 0;                         // particles placed so far
    for (int x = 0; x < n; x++)
        for (int y = 0; y < n; y++)
            for (int z = 0; z < n; z++) {
                    if (p < N) {
                        r[p][0] = (x + 0.5) * a;
                        r[p][1] = (y + 0.5) * a;
                        r[p][2] = (z + 0.5) * a;
                    }
                    ++p;
            }

    // initialize velocities
    for (int p = 0; p < N; p++)
        for (int i = 0; i < 3; i++)
            v[p][i] = vMax * (2 * rand() / double(RAND_MAX) - 1);
}
```



Newtoni mozgásegyenletek

EZT A RENDSZERT KELL INTEGRÁLNI!!!

az i. atom mozgásegyenlete:

$$\vec{a}_i(t) \equiv \frac{d\vec{v}_i(t)}{dt} = \frac{d^2\vec{r}_i(t)}{dt^2} = \frac{\vec{F}_i}{m}$$

erők számítása

$$\vec{F}_{i\leftarrow j} = -\vec{F}_{j\leftarrow i} = 12\left(\vec{r}_i - \vec{r}_j\right)\left[\left(\frac{1}{r}\right)^{-14} - \left(\frac{1}{r}\right)^{-8}\right]$$

$$\vec{F}_i = \sum_{\substack{j=1 \\ j\neq i}}^{N} \vec{F}_{i\leftarrow j}$$

```
void computeAccelerations() {

    for (int i = 0; i < N; i++)          // set all accelerations to zero
        for (int k = 0; k < 3; k++)
            a[i][k] = 0;

    for (int i = 0; i < N-1; i++)        // loop over all distinct pairs i,j
        for (int j = i+1; j < N; j++) {
            double rij[3];               // position of i relative to j
            double rSqd = 0;
            for (int k = 0; k < 3; k++) {
                rij[k] = r[i][k] - r[j][k];
                rSqd += rij[k] * rij[k];
            }
            double f = 12 * (pow(rSqd, -7) - pow(rSqd, -4));
            for (int k = 0; k < 3; k++) {
                 a[i][k] += rij[k] * f;
                 a[j][k] -= rij[k] * f;
            }
        }
}
```

Verlet integrálási algoritmusok

sebesség Verlet algoritmus:

$$\vec{r}_i(t+dt) = \vec{r}_i(t) + \vec{v}_i(t)\,dt + \frac{1}{2}\vec{a}_i(t)\,dt^2$$

$$\vec{v}_i(t+dt) = \vec{v}_i(t) + \frac{1}{2}\left[\vec{a}_i(t+dt) + \vec{a}_i(t)\right]dt$$

```
void velocityVerlet(double dt) {
    computeAccelerations();
    for (int i = 0; i < N; i++)
        for (int k = 0; k < 3; k++) {
            r[i][k] += v[i][k] * dt + 0.5 * a[i][k] * dt * dt;
            v[i][k] += 0.5 * a[i][k] * dt;
        }
    computeAccelerations();
    for (int i = 0; i < N; i++)
        for (int k = 0; k < 3; k++)
            v[i][k] += 0.5 * a[i][k] * dt;
}
```

a feladattól függően más integrálási algoritmusok is alkalmazhatók (lásd 3. és 8. előadás)

Egyszerű makroszkopikus mennyiségek mérése

általános szabály:

$$\langle A \rangle = \frac{1}{N_T} \sum_{k=1}^{N_T} A(k\,dt)$$

potenciális energia $\langle V(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_N) \rangle = \langle V \rangle$

mozgási energia $\left\langle \frac{1}{2} \sum_i m_i v_i^2 \right\rangle = \langle K \rangle$

teljes energia $\langle E \rangle = \langle K \rangle + \langle V \rangle = E = állandó$

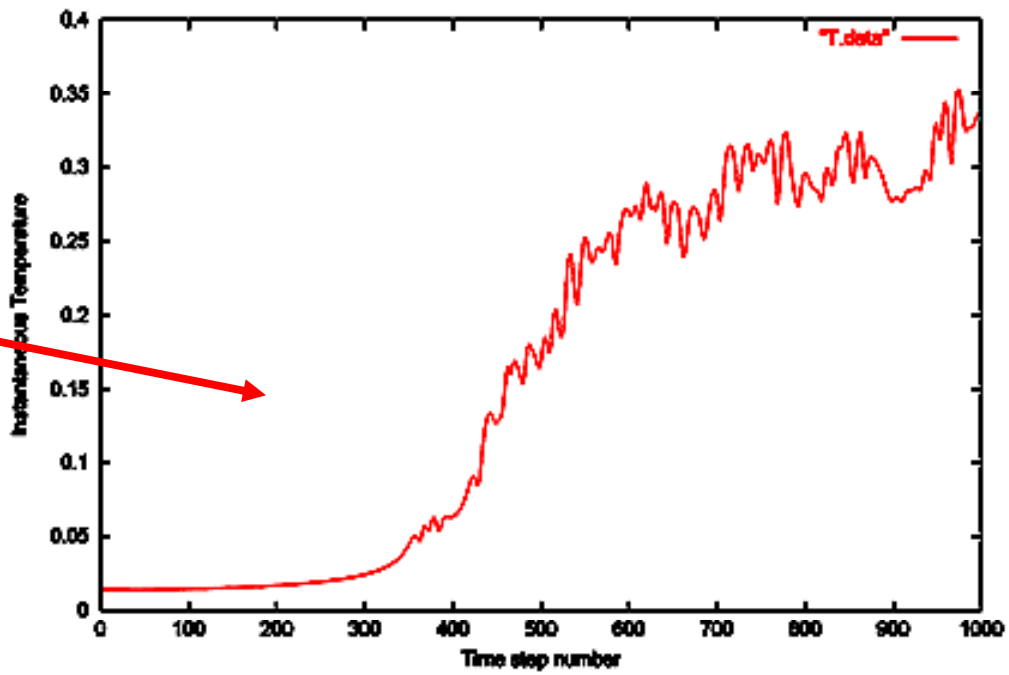
HŐMÉRSÉKLET: Boltzmann ekvipartíciós tétele alapján a pillanatnyi hőmérséklet

$$3(N-1)\frac{1}{2}k_B T = \left\langle \frac{m}{2} \sum_{i=1}^{N} v_2^2 \right\rangle$$

```
double instantaneousTemperature() {
    double sum = 0;
    for (int i = 0; i < N; i++)
        for (int k = 0; k < 3; k++)
            sum += v[i][k] * v[i][k];
    return sum / (3 * (N - 1));
}
```

A szimuláció

```
int main() {
    initialize();
    double dt = 0.01;
    ofstream file("T.data");
    for (int i = 0; i < 1000; i++) {
        velocityVerlet(dt);
        file << instantaneousTemperature() << '\n';
    }
    file.close();
}
```



HIÁNYOSSÁGOK:

- a térfogat nem állandó → határfeltételek alkalmazása
- a kezdeti pozíciókat és sebességeket jobban meg kell választani
  → LCK rács használata + Maxwell-Boltzmann eloszlás a sebességekre
- a rendszert kell hagyni, hogy termális egyensúlyba kerüljön az adott hőmérsékleten
- makroszkopikus mennyiségek átlagának mérése

Javított verzió

kezdeti pozíciók LCK inicializálása
sebességek inicializálása Maxwell-Boltzmann statisztika alapján
periodikus határfeltételek alkalmazása
minimum kép elve

paraméterek

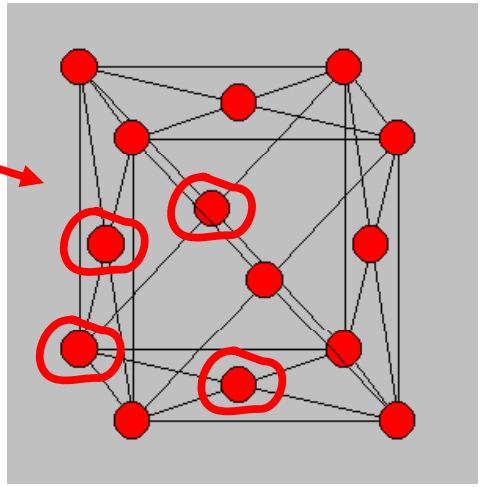
```
int N = 64;                 // number of particles
double rho = 1.2;           // density (number per unit volume)
double T = 1.0;             // temperature
double **r;                 // positions
double **v;                 // velocities
double **a;                 // accelerations
```

inicializálás

```
void initialize() {
    r = new double* [N];
    v = new double* [N];
    a = new double* [N];
    for (int i = 0; i < N; i++) {
        r[i] = new double [3];
        v[i] = new double [3];
        a[i] = new double [3];
    }
    initPositions();
    initVelocities();
}
```

Kezdeti pozíciók

lapcentrált köbös rácsra helyezzük az atomokat



(0,0,0), (0.5,0.5,0),
(0.5,0,0.5), (0,0.5,0.5)

```
double L;                          // linear size of cubical volume

void initPositions() {

    // compute side of cube from number of particles and number density
    L = pow(N / rho, 1.0/3);

    // find M large enough to fit N atoms on an fcc lattice
    int M = 1;
    while (4 * M * M * M < N)
        ++M;
    double a = L / M;              // lattice constant of conventional cell
    // 4 atomic positions in fcc unit cell
    double xCell[4] = {0.25, 0.75, 0.75, 0.25};
    double yCell[4] = {0.25, 0.75, 0.25, 0.75};
    double zCell[4] = {0.25, 0.25, 0.75, 0.75};
    int n = 0;                     // atoms placed so far
    for (int x = 0; x < M; x++)
        for (int y = 0; y < M; y++)
            for (int z = 0; z < M; z++)
                for (int k = 0; k < 4; k++)
                    if (n < N) {
                        r[n][0] = (x + xCell[k]) * a;
                        r[n][1] = (y + yCell[k]) * a;
                        r[n][2] = (z + zCell[k]) * a;
                        ++n;
                    }
}
```

Kezdeti sebességek

```
double gasdev () {
     static bool available = false;
     static double gset;
     double fac, rsq, v1, v2;
     if (!available) {
          do {
               v1 = 2.0 * rand() / double(RAND_MAX) - 1.0;
               v2 = 2.0 * rand() / double(RAND_MAX) - 1.0;
               rsq = v1 * v1 + v2 * v2;
          } while (rsq >= 1.0 || rsq == 0.0);
          fac = sqrt(-2.0 * log(rsq) / rsq);
          gset = v1 * fac;
          available = true;
          return v2*fac;
     } else {
          available = false;
          return gset;
     }
}
```

Box-Müller algoritmus

$$P(x) = \frac{e^{-(x-x_0)^2/(2\sigma^2)}}{\sqrt{2\pi\sigma^2}}$$

Maxwell-Boltzmann eloszlás

$$P(\mathbf{v}) = \left(\frac{m}{2\pi k_\mathrm{B} T}\right)^{3/2} e^{-m\left(v_x^2+v_y^2+v_z^2\right)/(2k_\mathrm{B}T)}$$

```
void initVelocities() {

    // Gaussian with unit variance
    for (int n = 0; n < N; n++)
        for (int i = 0; i < 3; i++)
            v[n][i] = gasdev();
    // Adjust velocities so center-of-mass velocity is zero
    double vCM[3] = {0, 0, 0};
    for (int n = 0; n < N; n++)
        for (int i = 0; i < 3; i++)
            vCM[i] += v[n][i];
    for (int i = 0; i < 3; i++)
        vCM[i] /= N;
    for (int n = 0; n < N; n++)
        for (int i = 0; i < 3; i++)
            v[n][i] -= vCM[i];

    // Rescale velocities to get the desired instantaneous temperature
    rescaleVelocities();
}
```

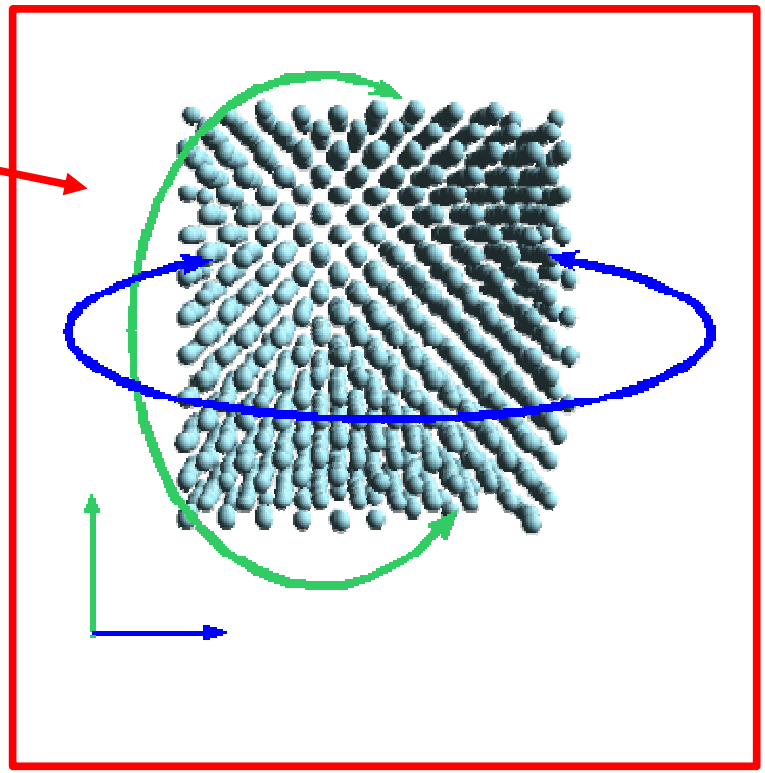
$$\mathbf{v}_\mathrm{CM} = \frac{\sum_{i=1}^{N} m\mathbf{v}_i}{\sum_{i=1}^{N} m}$$

```
void rescaleVelocities() {
    double vSqdSum = 0;
    for (int n = 0; n < N; n++)
        for (int i = 0; i < 3; i++)
            vSqdSum += v[n][i] * v[n][i];
    double lambda = sqrt( 3 * (N-1) * T / vSqdSum );
    for (int n = 0; n < N; n++)
        for (int i = 0; i < 3; i++)
            v[n][i] *= lambda;
}
```

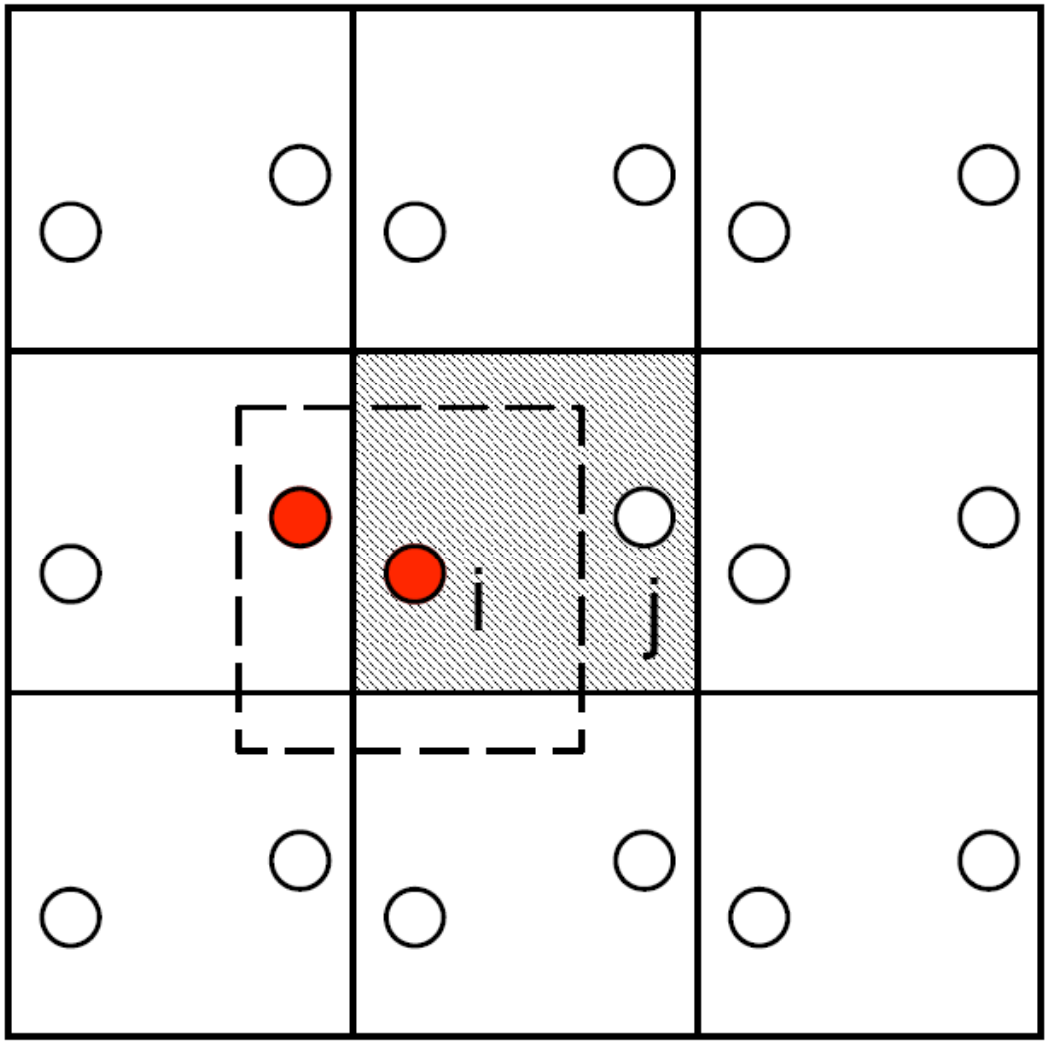
$$\mathbf{v}_i \longrightarrow \lambda\mathbf{v}_i \qquad \lambda = \sqrt{\frac{3(N-1)k_\mathrm{B}T}{\sum_{i=1}^{N} m v_i^2}}$$

Mozgásegyenletek integrálása

periodikus határfeltételek



minimális kép konvenció




```
HA |x_ij|>L/2:
      HA x_ij<0: x_ij=x_ij+L
      KÜLÖNBEN: x_ij=x_ij-L
...
```

Mozgásegyenletek integrálása

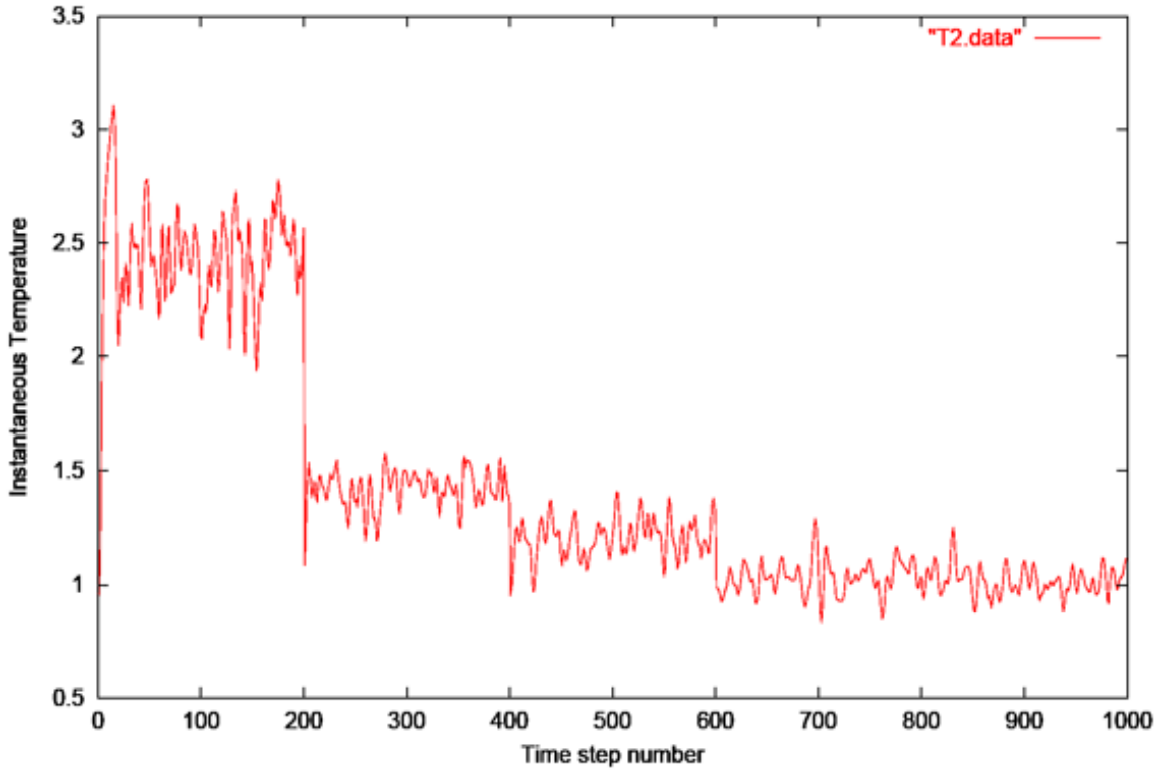
```
void computeAccelerations() {

    for (int i = 0; i < N; i++)          // set all accelerations to zero
        for (int k = 0; k < 3; k++)
            a[i][k] = 0;

    for (int i = 0; i < N-1; i++)        // loop over all distinct pairs i,j
        for (int j = i+1; j < N; j++) {
            double rij[3];               // position of i relative to j
            double rSqd = 0;
            for (int k = 0; k < 3; k++) {
                rij[k] = r[i][k] - r[j][k];
                // closest image convention
                if (abs(rij[k]) > 0.5 * L) {
                    if (rij[k] > 0)
                        rij[k] -= L;
                    else
                        rij[k] += L;
                }
                rSqd += rij[k] * rij[k];
            }
            double f = 12 * (pow(rSqd, -7) - pow(rSqd, -4));
            for (int k = 0; k < 3; k++) {
                 a[i][k] += rij[k] * f;
                 a[j][k] -= rij[k] * f;
            }
        }
}
```

Sebesség Verlet algoritmus

```
void velocityVerlet(double dt) {
    computeAccelerations();
    for (int i = 0; i < N; i++)
        for (int k = 0; k < 3; k++) {
            r[i][k] += v[i][k] * dt + 0.5 * a[i][k] * dt * dt;
            // use periodic boundary conditions
            if (r[i][k] < 0)
                r[i][k] += L;
            if (r[i][k] >= L)
                r[i][k] -= L;
            v[i][k] += 0.5 * a[i][k] * dt;
        }
    computeAccelerations();
    for (int i = 0; i < N; i++)
        for (int k = 0; k < 3; k++)
            v[i][k] += 0.5 * a[i][k] * dt;
}
```

A szimuláció

```
int main() {
    initialize();
    double dt = 0.01;
    ofstream file("T2.data");
    for (int i = 0; i < 1000; i++) {
        velocityVerlet(dt);
        file << instantaneousTemperature() << '\n';
        if (i % 200 == 0)
            rescaleVelocities();
    }
    file.close();
}
```



A nyomás meghatározása

tekintünk egy képzeletbeli egységnyi felületet a rendszerben:
a nyomás a felületre ható normális irányú erő nagyságához köthető
a felületen egységnyi idő alatt “áthaladó” impulzushoz köthető

viriál egyenlet alapján:

$$p(t) = \frac{N k_B T(t)}{V} + \frac{1}{DV} \sum_{i=1}^{N} \sum_{j>i} \vec{r}_{ij} \vec{F}_{ij}$$

a részecskék mozgásából származó járulék
az ideális gáztól jól ismert összefüggés

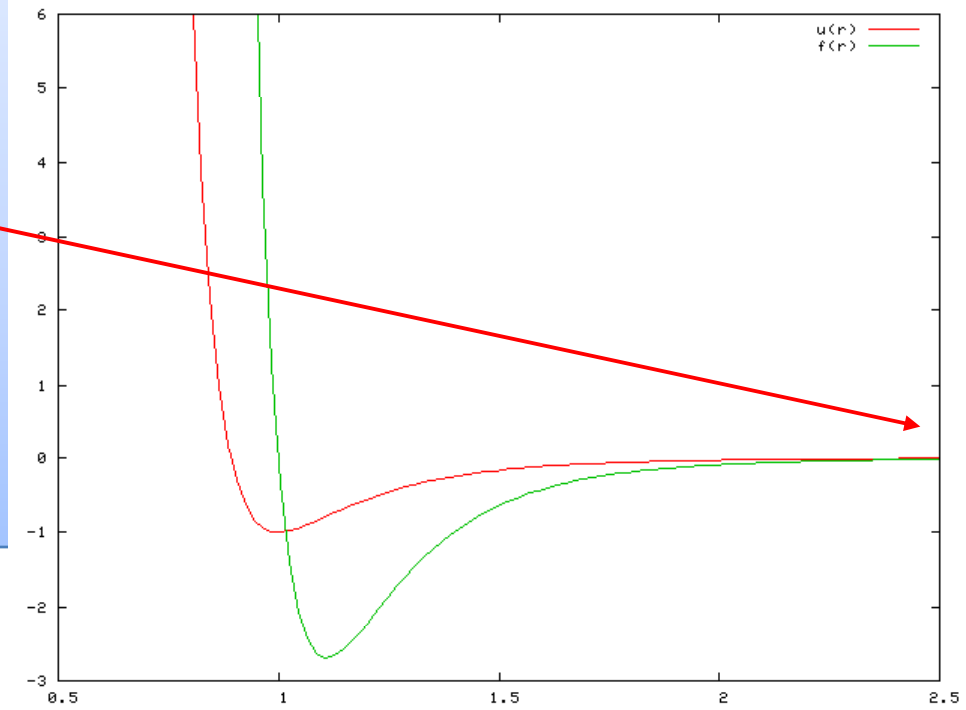
a felület különböző oldalán lévő részecskék között ható erők járuléka

MD kód optimalizálása

a legidőigényesebb rész az erők (gyorsulások) kiszámítása

L. Verlet, Phys. Rev. 159, 98 (1967)
két optimalizálást javasolt:

1. **a potenciál "farkának" levágása $r_{vág} = 2.5$ -nél**
2. **szomszédlisták tárolása:**
   - legyen $r_{max} > r_{vág}$ (pl. $r_{max} = 3.2$)
   - rögzítünk minden *(ij)* párt, melyre $r_{ij} < r_{max}$
   - a párok listáját csak minden 20-30 lépés után frissítjük



Másik lehetőség:
csatolt cellák módszere