Evaporation and soft-landing of a droplet of multicomponent suspension - an MD-like simulation using GPU

Author:
Tomasz Jakubczyk

CUDA simulation of droplet with nanoparticles

- Simulation is focused on interactions between nanoparticles and between nanoparticles and droplet surface that moves due to evaporation.
- CUDA parallel computing architecture greatly increases computation speed by executing simulation steps for all particles simultaneously on graphics card multiprocessors.

Some of simulation parameters

- Time step – manual or automatic
- Liquid viscosity
- Effective gravity
- Surface tension
- Scaling factor of Lennard-Jones potential
- Brown motion magnitude
- Coulomb repulsion
- Number of particles of different types
- Particle properties: mass, density, radius, charge and $\varepsilon_{i,j}$ between i-th and j-th particle type
- Evaporation rate
- Initial droplet radius

Algorithm main steps

- Arrange particles (by second indexing) according to their position into cells in order to identify which are close to each other and may interact.
- For particles that are in range of Lennard-Jones potential calculate force.
- For particles that interact with surface calculate force and keep it in the force table.
- Add force for effective gravity and force resulting from liquid viscosity.
- Calculate new velocity and position for all particles.

Simulation program usage

- To run simulation localize particles.exe and make sure that at your actual command line path particleType.cfg is present
- In particleType.cfg you can change particle types parameters
- Some of available command line options (where f is real/floating point number):
  -bigRadius0=f_1 – initial droplet radius in micrometers
  -timestep=f_2 – simulation time step; if not set it will dynamically change accordingly to simulation state to gain best possible performance without major numeric errors
  -damping=f_3 – liquid viscosity
  -gravity=f_4
  -A=f_5 - evaporation rate
  -brown=f_6 - Brown motion magnitude

Simulation GUI

- Right click – menu with options
- Left MB drag – change angle of view
- Scroll wheel – zoom
- Middle MB drag – move perspective
- H key (or in RMB menu) – show/hide sliders tweaking some simulation parameters

Formulas in code
Two particles collision in file particles_kernel_impl.cuh function collideSpheres

```
__device__
float3 collideSpheres(float3 posA, float3 posB,
                      float4 velA, float4 velB,
                      float radiusA, float radiusB,
                      float attraction)
{
    // calculate relative position
    float3 relPos = posB - posA;

    float dist = length(relPos);
    float collideDist = (radiusA + radiusB)*1.5f;//zasieg dzialania sil

    float3 force = make_float3(0.0f);

    if (dist < collideDist)
    {
        float3 norm = relPos / dist;

        float sigma=(params.particleRadius[(int)velA.w]+params.particleRadius[(int)velB.w])/(1.12246204f);
        float sd=sigma/dist;
        sd*=sd*sd*sd*sd*sd;
        float q1q2=params.normalizedCharge[(int)velA.w]*params.normalizedCharge[(int)velB.w];
        float a=!(velB.w<velA.w)?velA.w:velB.w;/**< min(x,y) */
        float b= (velA.w<velB.w)?velB.w:velA.w - a;/**< max(x,y)-min(x,y) */
        int epsilonIndex=(int)floor(a*(params.particleTypesNum-((a-1.0f)/2.0f))+b+0.5f);/**< wzór na index z zabezpieczeniem pr
        float epsilon=params.epsi*params.normalizeEpsilon[epsilonIndex];
        float foreScalar=48.0f*epsilon/dist*sd*(sd-0.5f)+attraction*q1q2/(dist*dist);
        force=-foreScalar*norm; //jest dobrze :-) Uwaga na kierunek wektora normalnego
////////////////////////////////////////////////////////////////////////////////
/*  tu wpisywac rownania na sily dla czastek bedacywch w zasiegu    */
////////////////////////////////////////////////////////////////////////////////
        float3 relVel=make_float3(velB)*norm-make_float3(velA)*norm;
        float relVelS=sqrt(relVel.x*relVel.x+relVel.y*relVel.y+relVel.z*relVel.z);
        float DtMax=0.1f*dist/relVelS;
        if(DtMax<0.00001f)
        {
            DtMax=0.00001f;
        }
        if(params.autoDt)
        {
            atomicMin(&globalDeltaTime,DtMax);
```

Formulas in code
Viscosity, gravity, interactions with surface, calculating new velocity and position in file particles_kernel_impl.cuh function integrate_functor::operator()(Tuple t)

```
            //vel+=vel*norm*0.1f*params.globalDamping;/**< na powierchni zmniejszone tłumienie w kierunku radialnym */

            forceF1=params.boundaryDamping*(abs(r0-R)-(params.particleRadius[(int)velData.w]));/**< siła napięcia powierzchnio
            force-=forceF1*norm;

        }
        /*else
        {
            force+=18.84955592f*params.viscosity*vel*params.particleRadius[(int)velData.w];
        }*/
        force-=18.84955592f*params.viscosity*params.globalDamping*vel*params.particleRadius[(int)velData.w];
        __syncthreads();
        if(params.calcSurfacePreasure && params.boundaries && r0>R-params.particleRadius[(int)velData.w] && r0<R+params.partic
        {
            float momentum=forceF1*globalDeltaTime;/**< pęd */
            momentum=abs(momentum);
            atomicAdd(&surfacePreasure,momentum);
        }
#endif

//dolna plaszczyzna

#if 1
        if (pos.y < -2.0f*params.bigradius0 + params.particleRadius[(int)velData.w])/**< blat */
        {
            pos.y = -2.0f*params.bigradius0 + params.particleRadius[(int)velData.w];
            vel.y = 0;
        }
#endif
        vel += params.gravity * globalDeltaTime;/**< grawitacja */
        //vel *= params.globalDamping;/**< lepkosc */
        pos += 0.5f * vel * globalDeltaTime;/**< przemieszczenie ze starą prędkością*/
        vel += force*(globalDeltaTime/params.particleMass[(int)velData.w]);/**< siły oddziaływań między cząsteczkami */
        // new position = old position + velocity * deltaTime
        pos += 0.5f * vel * globalDeltaTime;/**< przemieszczenie z nową prędkością*/

        // store new position and velocity
        thrust::get<0>(t) = make_float4(pos, posData.w);
        thrust::get<1>(t) = make_float4(vel, velData.w);
        thrust::get<2>(t) = make_float4(0.0f);/**< zerowanie tablicy sił na koniec kroku całkowania */
    }
```

Formulas in code
Droplet evaporation in file particles.cpp function parowanieKropliWCzasie

```
363 void parowanieKropliWCzasie()
364 {
365     //bigRadius=bigRadius0-A*sqrt(licznik*timestep);//r=r0-A*sqrt(t)
366     licznik++;
367     time_past+=timestep;
368     //std::cout<<psystem->getMaxParticleRadius()*pow(psystem->getNumParticles(),0.3f)<<std::endl;
369     if(bigRadius>(psystem->getMaxParticleRadius()*pow(psystem->getNumParticles(),0.3f))
370         && bigRadius>0.0f && hostSurfacePreasure<0.01f)
371     {
372         if(!koncowka_parowania)
373         {
374             SF=boundaryDamping;
375         }
376     }
377     else if(tSF<6.0f)
378     {
379         if(!koncowka_parowania)
380         {
381             koncowka_parowania=true;
382             time_to_end=(bigRadius0/A)*(bigRadius0/A)-time_past;
383         }
384         tSF+=6.0f*timestep*10.0f*A;///time_to_end;
385         boundaryDamping=SF*(-(tanh(tSF-3.0f)-1.0f)/2.0f);
386         if(boundaryDamping<0.0f)
387         {
388             boundaryDamping=0.0f;
389         }
390     }
391     else
392     {
393         boundaries=false;
394         boundaryDamping=0.0f;
395         //timestep=0.0001f;
396     }
397     if(bigRadius>0.0f)
398     {
399         psystem->setSurfaceVel((bigRadius-bigRadius0-A*sqrt(time_past))/timestep);
400         bigRadius=bigRadius0-A*sqrt(time_past);
401     }
402     else
```

References

- Code doxygen documentation: http://sigrond.github.io/particles/
- Code repository: https://github.com/sigrond/particles