

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 ϵ_{ij}
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=f₁ – initial droplet radius in micrometers
 - timestep=f₂ – simulation time step; if not set it will dynamically change accordingly to simulation state to gain best possible performance without major numeric errors
 - damping=f₃ – liquid viscosity
 - gravity=f₄
 - A=f₅ - evaporation rate
 - brown=f₆ - 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

```
378 __device__
379 float3 collideSpheres(float3 posA, float3 posB,
380                      float4 velA, float4 velB,
381                      float radiusA, float radiusB,
382                      float attraction)
383 {
384     // calculate relative position
385     float3 relPos = posB - posA;
386
387     float dist = length(relPos);
388     float collideDist = (radiusA + radiusB)*1.5f; //zasieg dzialania sil
389
390     float3 force = make_float3(0.0f);
391
392     if (dist < collideDist)
393     {
394         float3 norm = relPos / dist;
395
396         float sigma=(params.particleRadius[(int)velA.w]+params.particleRadius[(int)velB.w])/(1.12246204f);
397         float sd=sigma/dist;
398         sd*=sd*sd*sd*sd*sd;
399         float q1q2=params.normalizedCharge[(int)velA.w]*params.normalizedCharge[(int)velB.w];
400         float a!=(velB.w<velA.w)?velA.w:velB.w; /**< min(x,y) */
401         float b= (velA.w<velB.w)?velB.w:velA.w - a; /**< max(x,y)-min(x,y) */
402         int epsilonIndex=(int)floor(a*(params.particleTypesNum-((a-1.0f)/2.0f))+b+0.5f); /**< wzór na index z zabezpieczeniem p
403         float epsilon=params.epsi*params.normalizeEpsilon[epsilonIndex];
404         float foreScalar=48.0f*epsilon/dist*sd*(sd-0.5f)+attraction*q1q2/(dist*dist);
405         force=-foreScalar*norm; //jest dobrze :-> Uwaga na kierunek wektora normalnego
406         //////////////////////////////////////
407         /* tu wpisywac rownania na sily dla czastek bedacychw w zasiegu */
408         //////////////////////////////////////
409         float3 relVel=make_float3(velB)*norm-make_float3(velA)*norm;
410         float relVelS=sqrt(relVel.x*relVel.x+relVel.y*relVel.y+relVel.z*relVel.z);
411         float DtMax=0.1f*dist/relVelS;
412         if(DtMax<0.00001f)
413         {
414             DtMax=0.00001f;
415         }
416         if(params.autoDt)
417         {
418             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)

```
175 //vel+=vel*norm*0.1f*params.globalDamping;/**< na powierzchni zmniejszone tarcie w kierunku radialnym */
176
177 forceF1=params.boundaryDamping*(abs(r0-R)-(params.particleRadius[(int)velData.w]));/**< siła napięcia powierzchni
178 force-=forceF1*norm;
179
180 }
181 /*else
182 {
183     force+=18.84955592f*params.viscosity*vel*params.particleRadius[(int)velData.w];
184 }*/
185 force-=18.84955592f*params.viscosity*params.globalDamping*vel*params.particleRadius[(int)velData.w];
186 __syncthreads();
187 if(params.calcSurfacePressure && params.boundaries && r0>R-params.particleRadius[(int)velData.w] && r0<R+params.particleRadius[(int)velData.w])
188 {
189     float momentum=forceF1*globalDeltaTime;/**< pęd */
190     momentum=abs(momentum);
191     atomicAdd(&surfacePressure,momentum);
192 }
193 #endif
194
195 //dolna płaszczyzna
196
197 #if 1
198 if (pos.y < -2.0f*params.bigradius0 + params.particleRadius[(int)velData.w])/**< blat */
199 {
200     pos.y = -2.0f*params.bigradius0 + params.particleRadius[(int)velData.w];
201     vel.y = 0;
202 }
203 #endif
204 vel += params.gravity * globalDeltaTime;/**< grawitacja */
205 //vel *= params.globalDamping;/**< lepkość */
206 pos += 0.5f * vel * globalDeltaTime;/**< przemieszczenie ze starą prędkością*/
207 vel += force*(globalDeltaTime/params.particleMass[(int)velData.w]);/**< siły oddziaływań między cząsteczkami */
208 // new position = old position + velocity * deltaTime
209 pos += 0.5f * vel * globalDeltaTime;/**< przemieszczenie z nową prędkością*/
210
211 // store new position and velocity
212 thrust::get<0>(t) = make_float4(pos, posData.w);
213 thrust::get<1>(t) = make_float4(vel, velData.w);
214 thrust::get<2>(t) = make_float4(0.0f);/**< zerowanie tablicy sił na koniec kroku całkowania */
215 }
```


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(pssystem->getNumParticles(),0.3f)<<std::endl;
369     if(bigRadius>(psystem->getMaxParticleRadius()*pow(psystem->getNumParticles(),0.3f))
370         && bigRadius>0.0f && hostSurfacePressure<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>