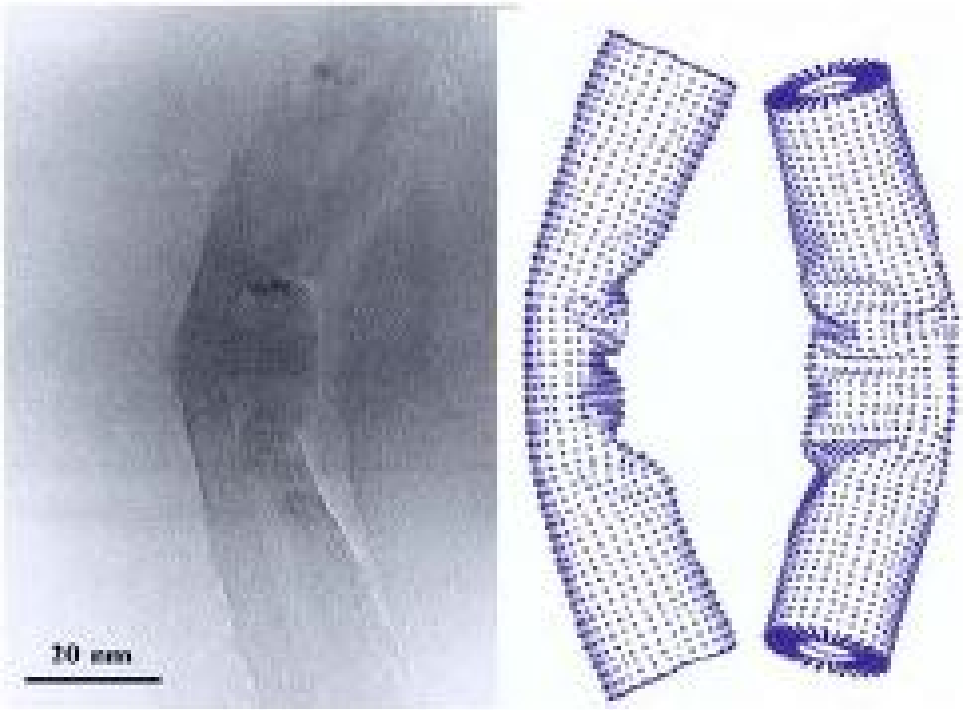
Particle Simulation using CUDA:

[1]Edgar Andrés Patiño Nariño
Departamento de Mecânica Computacional – DMC
Faculdade de Engenharia Mecânica – FEM
Universidade Estadual de Campinas - UNICAMP
[1]eapatinon@fem.unicamp.br

Introdução

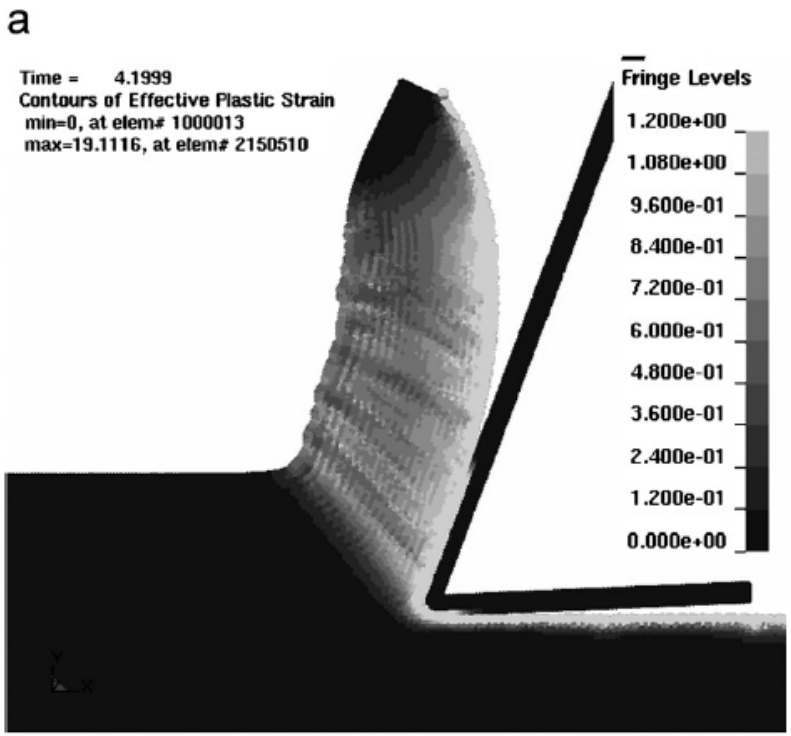
Os métodos de partículas [1] são técnicas comumente usadas na simulação de problemas físicos [2].



Observação experimental de encurvatura de nanotubos de carbono de paredes múltiplas (b) Simulacao de flambagem padrão MD em 2D. (c) Flambagem padrão da nanoestrutura MD em 3D [2].

Introdução

Os métodos de partículas [1] são técnicas comumente usadas na simulação de problemas físicos [2].



Limido, 2009, Usinagem Ortogonal SPH [3].

Introdução

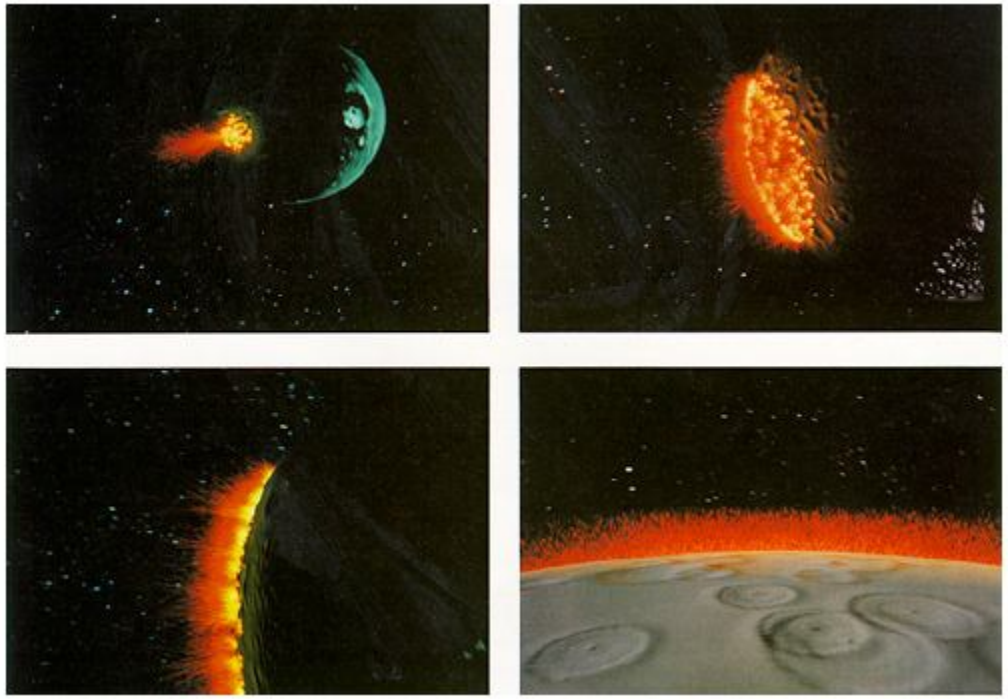
Os métodos de partículas [1] são técnicas comumente usadas na simulação de problemas físicos [2].

(a) Experiment (b) Simulation

Comparação experimental e resultado de simulação de uma lâmina de borracha em deflexão por uma coluna de água [2].

Introdução

Os métodos de partículas [1] são técnicas comumente usadas na simulação de problemas físicos [2].



Stark Trek (Jornada nas estrelas): The Wrath of Khan

"Particle system: A technique for modeling a class of fuzzy objects" *Rees 1983* [4].

Simulação de fluidos baseados em partículas

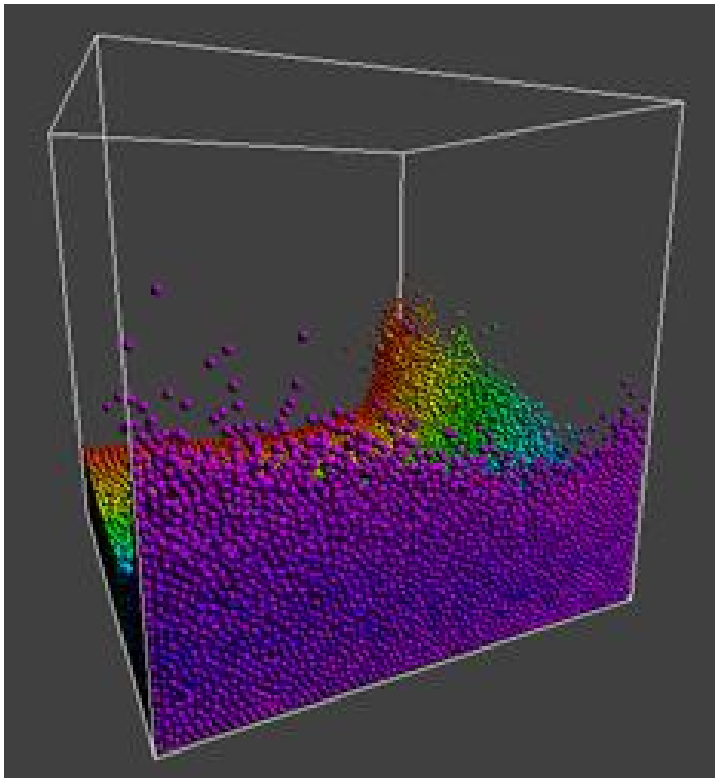
Vantagem

- A conservacao de massa é trivial.
- Fácil aplicação no problema de superfície livre.
- Apenas executa cálculos quando é necessário.
- Não precisa de malha.
- Fácil de paralelizar.

Desvantagem

- Precisa de um longo número de partículas para resultados realistas .

Particle Simulation using CUDA



http://www.youtube.com/watch?v=RqduA7myZok

Particle Simulation using CUDA

```
102 // create a color ramp
103 void colorRamp(float t, float *r)
104 {
105     const int ncolors = 7;
106     float c[ncolors][3] =
107     {
108
109         { 1.0, 0.0, 0.0, },
110         { 1.0, 0.4, 0.0, },
111         { 1.0, 1.0, 0.0, },
112         { 0.0, 1.0, 0.0, },
113         { 0.0, 1.0, 1.0, },
114         { 1.0, 0.0, 1.0, },
115         { 1.0, 0.0, 0.0, },
116     };
117     t = t * (ncolors-1);
118     int i = (int) t;
119     float u = t - floor(t);
120     r[0] = lerp(c[i][2], c[i+1][2], u);
121     r[1] = lerp(c[i][1], c[i+1][1], u);
122     r[2] = lerp(c[i][0], c[i+1][0], u);
123 }
```

```
180             float t = i / (float) m_numParticles;
181 #if 0
182             *ptr++ = rand() / (float) RAND_MAX;
183             *ptr++ = rand() / (float) RAND_MAX;
184             *ptr++ = rand() / (float) RAND_MAX;
185 #else
186             colorRamp(t, ptr);
187             ptr+=3;
188 #endif
189             *ptr++ = 1.0f;
190         }
191
192         glUnmapBufferARB(GL_ARRAY_BUFFER);
193     }
194     else
195     {
196         checkCudaErrors(cudaMalloc((void **)&m_cudaColor
197     }
198
199     sdkCreateTimer(&m_timer);
200
201     setParameters(&m_params);
202
203     m_bInitialized = true;
204 }
```

Particle Simulation using CUDA

```
238 // step the simulation
239 void
240 ParticleSystem::update(float deltaTime)
241 {
242     assert(m_bInitialized);
243 
244     float *dPos;
245 
246     if (m_bUseOpenGL)
247     {
248         dPos = (float *) mapGLBufferObject(&m_cuda_posvbo_resource);
249     }
250     else
251     {
252         dPos = (float *) m_cudaPosVBO;
253     }
254 
255     // update constants
256     setParameters(&m_params);
257 
258     // integrate
259     integrateSystem(
260         dPos,
261         m_dVel,
262         deltaTime,
263         m_numParticles);
264 
265     // calculate grid hash
266     calcHash(
267         m_dGridParticleHash,
268         m_dGridParticleIndex,
269         dPos,
270         m_numParticles);
271 
272     // sort particles based on hash
273     sortParticles(m_dGridParticleHash, m_dGridParticleIndex, m_numParticles);
```

Funções de transferência

Desejável: Distinguir variações abruptas, exibir as transições entre regiões de forma suave e permitir modificações em tempo real.

Dados de interesse

↓

Características óticas →

Cores, Absorção e emissão

http://www.youtube.com/watch?v=3tkj7ImCjkc

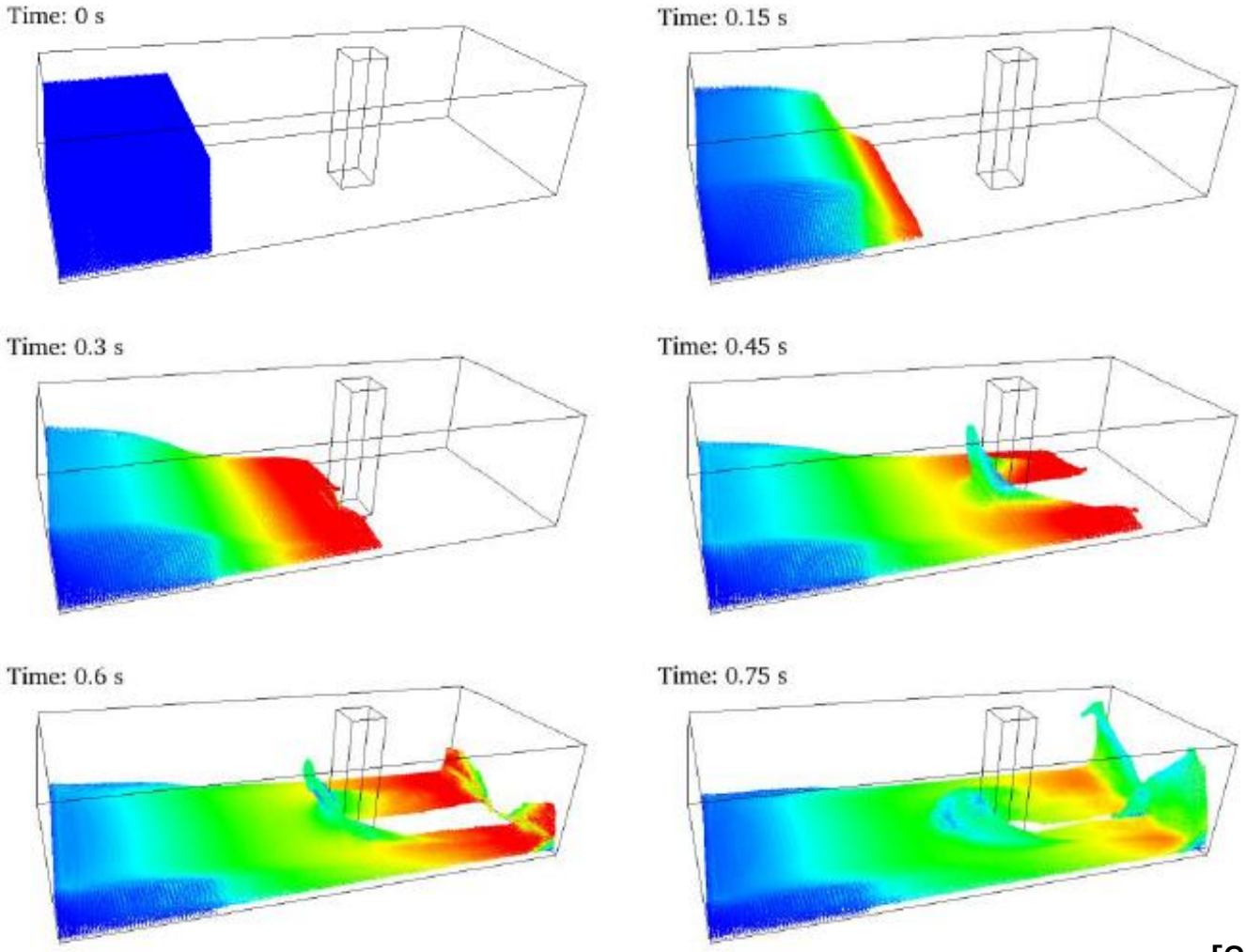
http://www.automotiveproductsfinder.com/APFCONTENT/articles/cfd-comes-of-age.php

Proposta de Projeto do curso

Modificar o código em CUDA "Particle Simulation using CUDA":

- Uso de ***função de transferência***, para distinguir as velocidades das partículas em tempo real.
- Distinguir variações abruptas em uma ***faixa de partículas.***

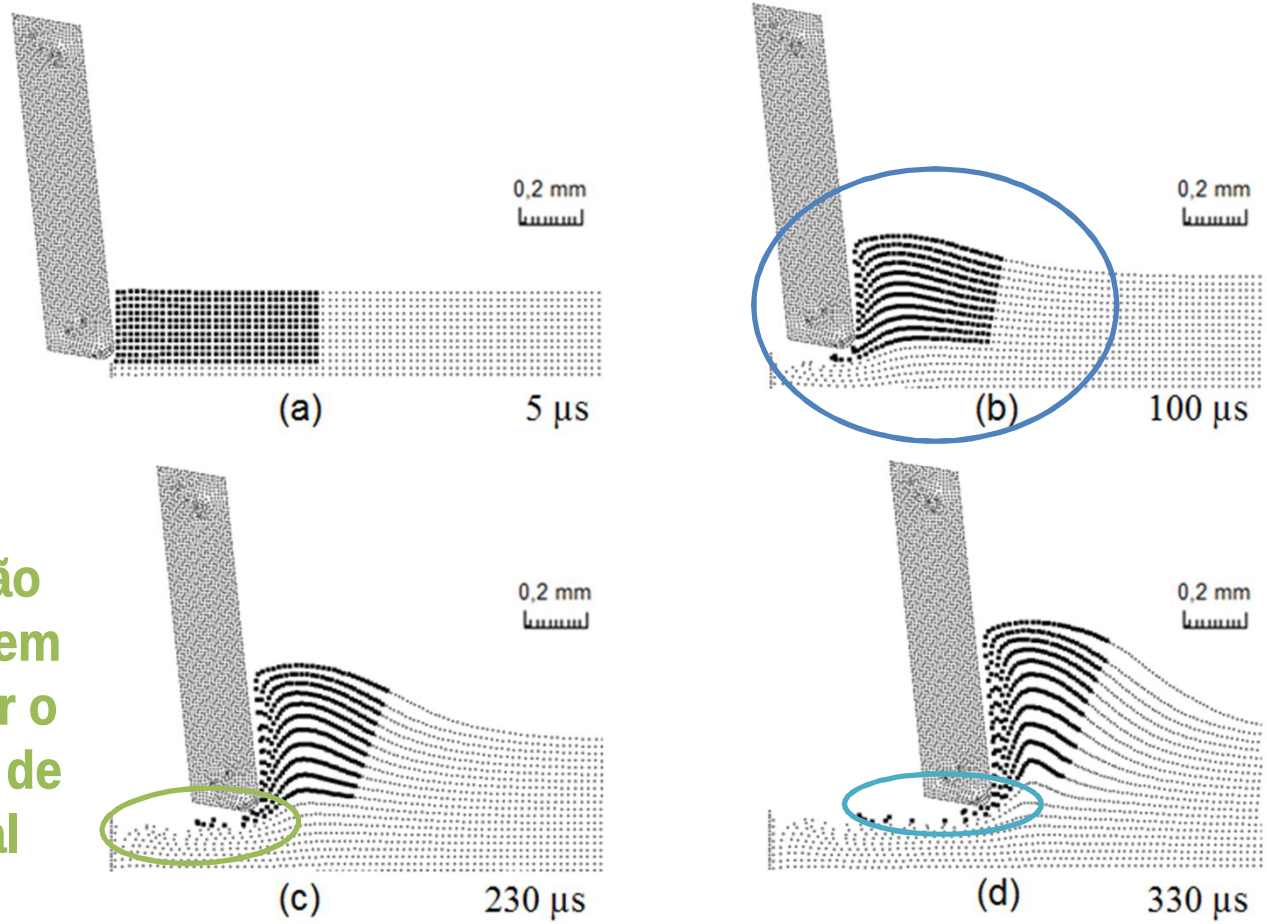
Proposta de Projeto do curso



[3,4]

Proposta de Projeto do curso

Remoção de Material e superfície gerada:



Penetração da ferramenta sobre o material de trabalho

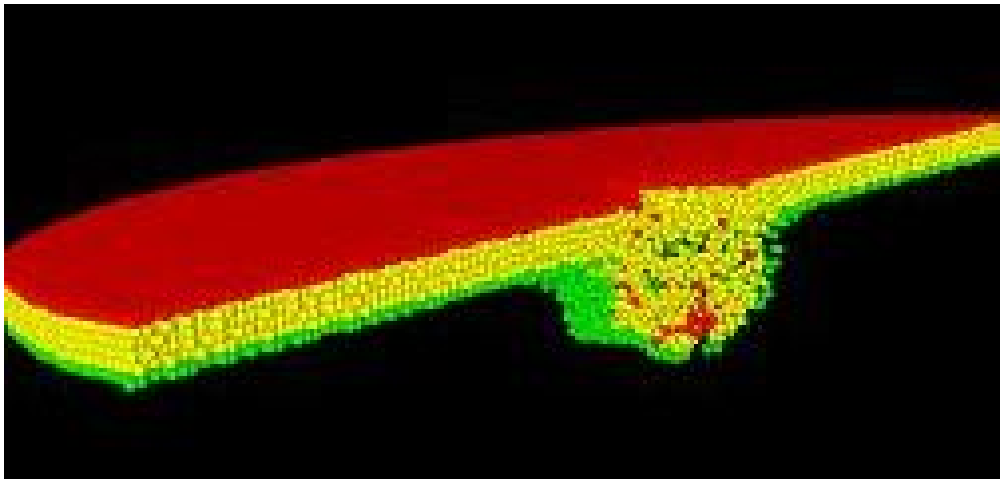
Separação Natural sem predefinir o arranque de material

Ponto de estancamento

Proposta de Projeto do curso

Modificar o código em CUDA “Particle Simulation using CUDA”:

- Gerar corte, para visualização das partículas, em posições intermediárias .

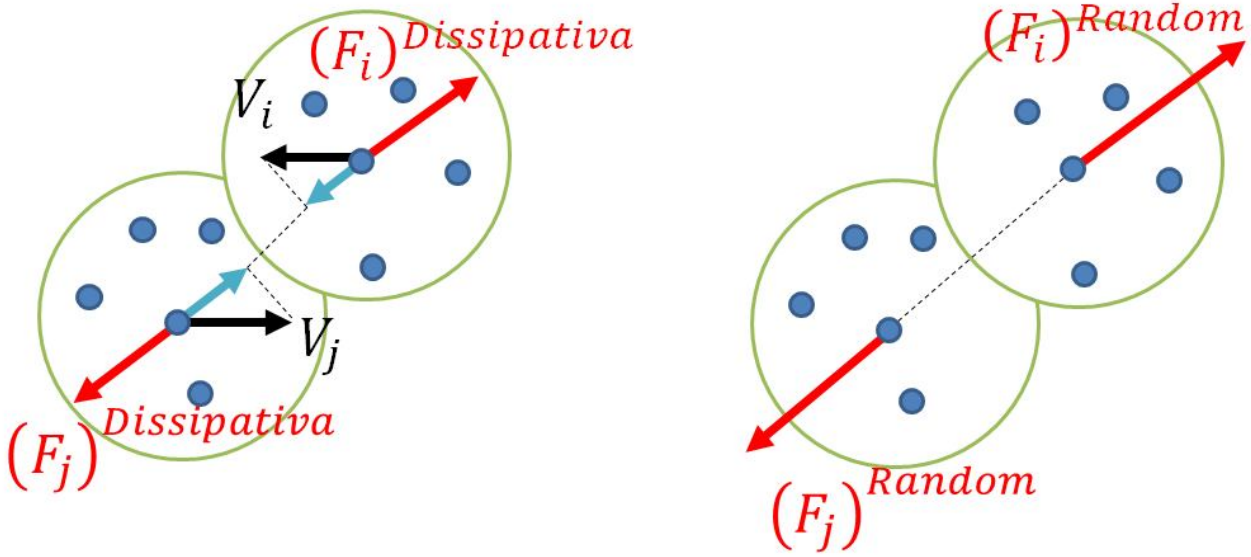


Proposta de Projeto do curso

Modificar o código em CUDA "Particle Simulation using CUDA":

- Para o processamento de colisões entre partículas mudando do *Discrete Element Methods* (**DEM**) [5] para o método *Dissipative Particle Dynamics* (**DPD**) [6].

Dissipative Particle Dynamics



A **força Dissipativa** (atrito) reduze a relação entre pares de partículas.
A **força Aleatória** (Randon) compensa os graus de liberdade eliminados.
As **forças Dissipativas e Aleatórias** formam o termostato do DPD.
As grandezas das **forças Dissipativas e Aleatórias** são definidas pelo teorema de **Flutuação-dissipação**.

Dissipative Particle Dynamics

$$\mathrm{d}\mathbf{r}_i = \mathbf{v}_i \mathrm{d}t$$

$$m_i \mathrm{d}\mathbf{v}_i = \sum_{i \neq j} \mathbf{F}_{ij}^{(C)} \left(r_{ij} \right) \vec{e}_{ij} \mathrm{d}t - \gamma \sum_{i \neq j} \omega^D \left(r_{ij} \right) \left(\mathbf{v}_{ij} \cdot \vec{e}_{ij} \right) \vec{e}_{ij} \mathrm{d}t$$

$$+ \sigma \sum_{i \neq j} \omega^R \left(r_{ij} \right) \xi_{ij} \vec{e}_{ij} \mathrm{dW}_{\mathrm{ij}}$$

Atualização temporal por meio de ***Verlet Algorithm***

Para velocidade V_i e Posicao X_i para tempo

$$t_{n+1} = t_n + \Delta t$$

UNICAMP

Obrigado pela sua atenção

Perguntas?

Edgar Andres Patiño Nariño
(eapatinon@fem.unicamp.br
)

REFERÊNCIAS

[1] S. Li, W. K. Liu, and Shaofan Li ·Wing Kam Liu, *Meshfree particle method*, vol. 25, no. 2–3. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 289–296 vol.1.

[2] G. R. Liu, “Mesh Free Methods: Moving Beyond the Finite Element Method,” *Applied Mechanics Reviews*, vol. 56, no. 2, p. B17, 2003.

[3] J. Limido, C. Espinosa, M. Salau, and J. L. Lacome, “SPH method applied to high speed cutting modelling,” *International Journal of Mechanical Sciences*, vol. 49, pp. 898–908, 2007.

[4] Ericson, C. (2004). Real-time collision detection. *Chemistry & …*. Retrieved from http://onlinelibrary.wiley.com/doi/10.1002/cbdv.200490137/abstract

[5] A. A. Munjiza, *The Combined Finite-Discrete Element Method*. Wiley, 2004, p. 348.

[6] M. Revenga, I. Zúñiga, and P. Español, “Boundary conditions in dissipative particle dynamics,” *Computer Physics Communications*, vol. 121–122, no. null, pp. 309–311, Sep. 1999.

REFERÊNCIAS

[7] M. Griebel, S. Knapek, and G. Zumbusch, *Numerical simulation in molecular dynamics*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2007.

[8] Ericson, C. (2004). Real-time collision detection. *Chemistry & …*. Retrieved from http://onlinelibrary.wiley.com/doi/10.1002/cbdv.200490137/abstract.

[9] Satish, N., Sciences, C., Harris, M., Garland, M., & Clara, S. (2009). Designing Efficient Sorting Algorithms for Manycore GPUs, (May), 1–10.

[10] Nguyen, H. (2007). *Gpu gems 3*. Retrieved from http://dl.acm.org/citation.cfm?id=1407436

[11] Hou, Q., Zhou, K., & Guo, B. (2008). BSGP: bulk-synchronous GPU programming. *ACM Transactions on Graphics (TOG)*. Retrieved from http://dl.acm.org/citation.cfm?id=1360618