PARPING

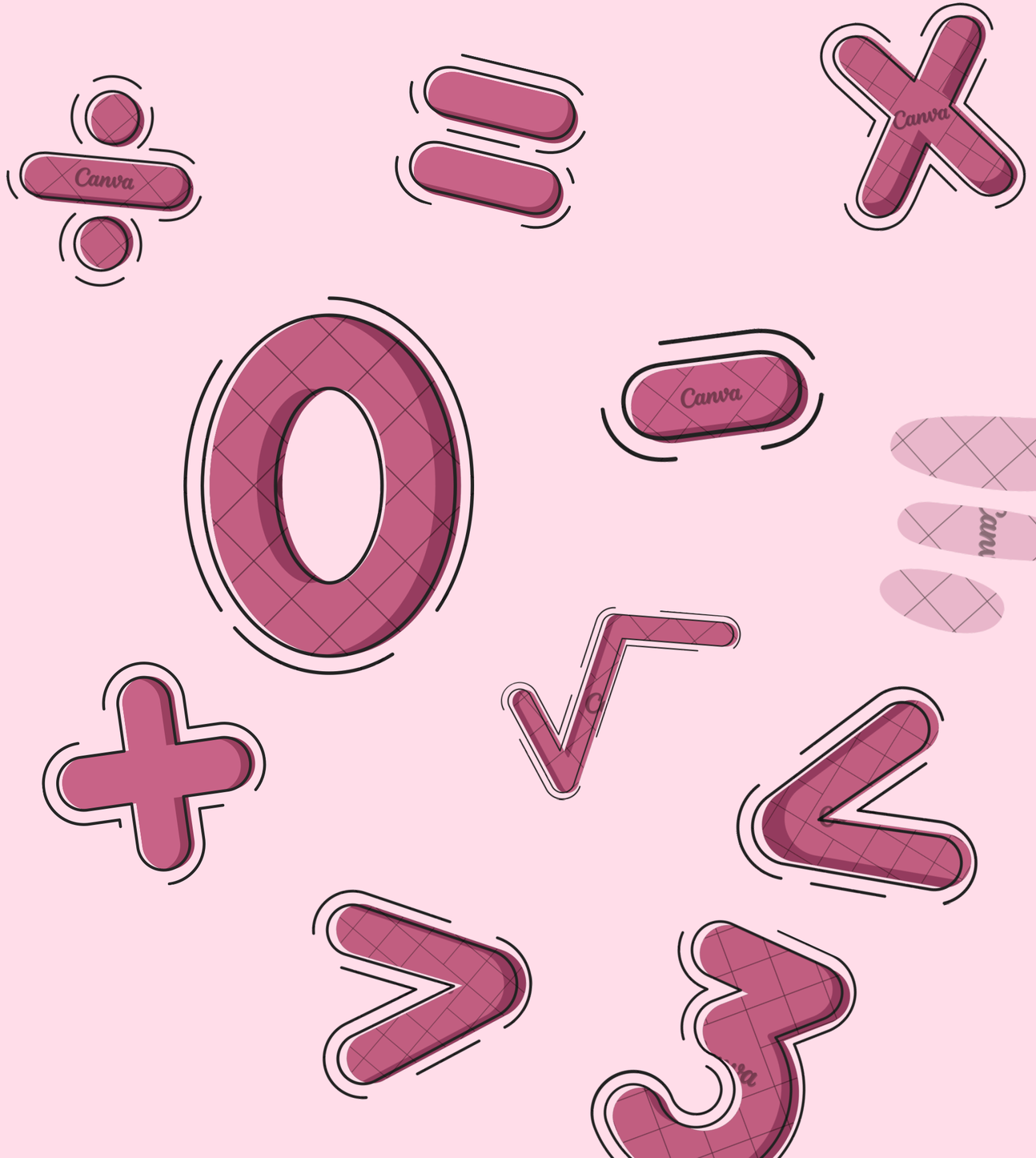
PROJET GALAX

Florian Huynh, Tchadel Icard

Introduction

Galax est une simulation de collision entre deux galaxies basée sur le problème des N-corps

- Objectif : maximer le nombre de FPS pour 10,000 particules sans affichage
- Techniques : OpenMD, SIMD (AVX2), FMA



Implémentation naïve

- Complexité algorithmique : $O(n^2)$
- Objectif : utiliser cette implémentation comme base de comparaison

- Résultats
  - 2000 particules → 50 FPS
  - 5000 particules → 8.4 FPS
  - 10000 particules → 2 FPS

```
11  void Model_CPU_naive
12  ::step()
13  {
14      std::fill(accelerationsx.begin(), accelerationsx.end(), 0);
15      std::fill(accelerationsy.begin(), accelerationsy.end(), 0);
16      std::fill(accelerationsz.begin(), accelerationsz.end(), 0);
17
18      for (int i = 0; i < n_particles; i++)
19      {
20          for (int j = 0; j < n_particles; j++)      Mathieu Léonardon, 4 years ago • Initial co
21          {
22              if(i != j)
23              {
24                  const float diffx = particles.x[j] - particles.x[i];
25                  const float diffy = particles.y[j] - particles.y[i];
26                  const float diffz = particles.z[j] - particles.z[i];
27
28                  float dij = diffx * diffx + diffy * diffy + diffz * diffz;
29
30                  if (dij < 1.0)
31                  {
32                      dij = 10.0;
33                  }
34                  else
35                  {
36                      dij = std::sqrt(dij);
37                      dij = 10.0 / (dij * dij * dij);
38                  }
39
40                  accelerationsx[i] += diffx * dij * initstate.masses[j];
41                  accelerationsy[i] += diffy * dij * initstate.masses[j];
42                  accelerationsz[i] += diffz * dij * initstate.masses[j];
43              }
44          }
45      }
46
47      for (int i = 0; i < n_particles; i++)
48      {
49          velocitiesx[i] += accelerationsx[i] * 2.0f;
50          velocitiesy[i] += accelerationsy[i] * 2.0f;
51          velocitiesz[i] += accelerationsz[i] * 2.0f;
52          particles.x[i] += velocitiesx   [i] * 0.1f;
53          particles.y[i] += velocitiesy   [i] * 0.1f;
54          particles.z[i] += velocitiesz   [i] * 0.1f;
55      }
56  }
```

Implémentation CPU Fast avec OpenMP

- Utilisation d'OpenMP pour pouvoir faire tourner l'algorithme de manière parallèle (4 coeurs de la machine)
- Accélération attendue de 4

```
23  void Model_CPU_fast
24  ::step()
25  {
26      std::fill(accelerationsx.begin(), accelerationsx.end(), 0);
27      std::fill(accelerationsy.begin(), accelerationsy.end(), 0);
28      std::fill(accelerationsz.begin(), accelerationsz.end(), 0);
29
30      #pragma omp parallel for schedule(static)
31      for (int i = 0; i < n_particles; i++)
32      {
33          for (int j = 0; j < n_particles; j++)
34          {
35              if(i != j)
36              {
37                  const float diffx = particles.x[j] - particles.x[i];
38                  const float diffy = particles.y[j] - particles.y[i];
39                  const float diffz = particles.z[j] - particles.z[i];
40
41                  float dij = diffx * diffx + diffy * diffy + diffz * diffz;
42
43                  if (dij < 1.0)
44                  {
45                      dij = 10.0;
46                  }
47                  else
48                  {
49                      dij = std::sqrt(dij);
50                      dij = 10.0 / (dij * dij * dij);
51                  }
52
53                  accelerationsx[i] += diffx * dij * initstate.masses[j];
54                  accelerationsy[i] += diffy * dij * initstate.masses[j];
55                  accelerationsz[i] += diffz * dij * initstate.masses[j];
56              }
57          }
58      }
```

Implémentation CPU Fast avec OpenMP

CPU	FPS	Particules	Accélération
CPU Naïve	50	2000	REF
CPU Naïve	8.4	5000	REF
CPU Naïve	2	10000	REF
CPU Fast (OpenMP)	150	2000	3
CPU Fast (OpenMP)	20	5000	2,38
CPU Fast (OpenMP)	5,6	10000	2,8

Résultat inférieur à l’accélération attendue

Implémentation CPU Fast avec OpenMP + SIMD V1

- Utilisation de SIMD pour exécuter une instruction sur plusieurs données en parallèle, accélérant les calculs.
- Accélération attendue de 8

```
#pragma omp parallel for schedule(static)
for (int i = 0; i < n_particles; i++)
{
    b_type acc_x = b_type(0.0f);
    b_type acc_y = b_type(0.0f);
    b_type acc_z = b_type(0.0f);
    for (int j = 0; j < n_particles; j += b_type::size)
    {
        if(i != j)
        {
            b_type xj = b_type::load_unaligned(&particles.x[j]);
            b_type yj = b_type::load_unaligned(&particles.y[j]);
            b_type zj = b_type::load_unaligned(&particles.z[j]);
            b_type mj = b_type::load_unaligned(&initstate.masses[j]);

            b_type xi = b_type(particles.x[i]);
            b_type yi = b_type(particles.y[i]);
            b_type zi = b_type(particles.z[i]);

            b_type diffx = xj - xi;
            b_type diffy = yj - yi;
            b_type diffz = zj - zi;

            b_type dij = diffx * diffx + diffy * diffy + diffz * diffz;

            // Compute inverse squared-root and handle singularities
            auto mask = dij < b_type(1.0f);
            dij = xs::select(mask, b_type(10.0f), xs::rsqrt(dij) * xs::rsqrt(dij) * xs::rsqrt(dij) * b_type(10.0f));


            acc_x += diffx * dij * mj;
            acc_y += diffy * dij * mj;
            acc_z += diffz * dij * mj;

        }
    }
    // Store results back
    accelerationsx[i] += xs::reduce_add(acc_x);
    accelerationsy[i] += xs::reduce_add(acc_y);
    accelerationsz[i] += xs::reduce_add(acc_z);
}
```

Implémentation CPU Fast avec OpenMP + SIMD V2

- Nouvelle version avec les constantes qui sont définies en haut
- Remplacement du select par un max (moins couteux en calculs)
- On calcule qu'une seule fois la racine inverse au lieu de 3 fois avant

```
227     const b_type b10 = b_type(10.0f);
228     const b_type b1 = b_type(1.0f);
229
230     #pragma omp parallel for schedule(static)
231     for (int i = 0; i < n_particles; i++)
232     {
233         b_type acc_x = b_type(0.0f);
234         b_type acc_y = b_type(0.0f);
235         b_type acc_z = b_type(0.0f);
236
237         b_type xi = b_type(particles.x[i]);
238         b_type yi = b_type(particles.y[i]);
239         b_type zi = b_type(particles.z[i]);
240         for (int j = 0; j < n_particles; j += b_type::size)
241         {
242             if (i != j) {
243                 b_type xj = b_type::load_unaligned(&particles.x[j]);
244                 b_type yj = b_type::load_unaligned(&particles.y[j]);
245                 b_type zj = b_type::load_unaligned(&particles.z[j]);
246                 b_type mj = b_type::load_unaligned(&initstate.masses[j]);
247
248
249                 b_type diffx = xj - xi;
250                 b_type diffy = yj - yi;
251                 b_type diffz = zj - zi;
252
253                 b_type dij = diffx * diffx + diffy * diffy + diffz * diffz;
254                 dij = xs::max(dij, b1);
255
256                 // Compute inverse squared-root and handle singularities
257                 b_type inv_dij = xs::rsqrt(dij);
258                 inv_dij = inv_dij * inv_dij * inv_dij;
259
260                 dij = b10 * inv_dij;
261
262
263                 acc_x += diffx * dij * mj;
264                 acc_y += diffy * dij * mj;
265                 acc_z += diffz * dij * mj;
266             }
267         }
268         // Store results back
269         accelerationsx[i] += xs::reduce_add(acc_x);
270         accelerationsy[i] += xs::reduce_add(acc_y);
271         accelerationsz[i] += xs::reduce_add(acc_z);
272     }
```

Implémentation CPU Fast avec OpenMP + SIMD V3

- On charge 8 particules à la fois et on calcule toutes les interactions avec un rotate
- Permet d’optimiser les accès mémoire
- Elle ne réduit pas le nombre d’opératations effectuées

```
418     const b_type b10 = b_type(10.0f);
419     const b_type b1 = b_type(1.0f);
420
421     #pragma omp parallel for
422     for (int i = 0; i < n_particles; i += b_type::size)
423     {
424         b_type xi = b_type::load_unaligned(&particles.x[i]);
425         b_type yi = b_type::load_unaligned(&particles.y[i]);
426         b_type zi = b_type::load_unaligned(&particles.z[i]);
427
428         b_type acc_x = b_type(0.0f);
429         b_type acc_y = b_type(0.0f);
430         b_type acc_z = b_type(0.0f);
431         for (int j = 0; j < n_particles; j += b_type::size)
432         {
433             b_type xj = b_type::load_unaligned(&particles.x[j]);
434             b_type yj = b_type::load_unaligned(&particles.y[j]);
435             b_type zj = b_type::load_unaligned(&particles.z[j]);
436             b_type mj = b_type::load_unaligned(&initstate.masses[j]);
437
438             for (int k = 0; k < b_type::size; k++) {
439                 b_type diffx = xj - xi;
440                 b_type diffy = yj - yi;
441                 b_type diffz = zj - zi;
442
443                 b_type dij = diffx * diffx + diffy * diffy + diffz * diffz;
444                 dij = xs::max(dij, b1);
445
446                 // Compute inverse squared-root and handle singularities
447                 b_type inv_dij = xs::rsqrt(dij);
448                 inv_dij = inv_dij * inv_dij * inv_dij;
449
450                 dij = b10 * inv_dij;
451
452
453                 acc_x += diffx * dij * mj;
454                 acc_y += diffy * dij * mj;
455                 acc_z += diffz * dij * mj;
456
457                 xj = xs::rotate_right<1>(xj);
458                 yj = xs::rotate_right<1>(yj);
459                 zj = xs::rotate_right<1>(zj);
460                 mj = xs::rotate_right<1>(mj);
461             }
462         }
463         acc_x.store_unaligned(&accelerationsx[i]);
464         acc_y.store_unaligned(&accelerationsy[i]);
465         acc_z.store_unaligned(&accelerationsz[i]);
466     }
```

Implémentation CPU Fast avec OpenMP

CPU	FPS	Particules	Accélération
CPU Naïve	2	10000	REF
CPU Fast V0	5,6	10000	2,8 (CPU_Naive)
CPU Fast V1	65	10000	32,5 (CPU_Naive) 11,6 (CPU_Fast)
CPU Fast V2	84	10000	42 (CPU_Naive) 15 (CPU_Fast)
CPU Fast V3	90 (110 sans GNOME)	10000	45 (CPU_Naive) 16 (CPU_Fast)

Accélération attendue liée à SIMD de 8

Optimisation effectuée par le compilateur ?

Mesures avec OpenMP pour 10 000 particules

- OpenMP avec 1 thread : ~1,5 FPS
- OpenMP avec 4 threads : ~5,2-6 FPS

Pour comparer avec la version naïve

- CPU Naïve avec 10 000 particules : ~2,1 FPS

On a bien une accélération de 4 avec le même code

Validate

```
f22huynh@fl-tp-br-633:~/galax_eleves/build$ ./bin/galax -c CPU_FAST -n 10000 --display NO --validate
State updates per second: 0 ;               average distance vs reference: 1.94313e-05; min error : 0; max error : 0.000111808
State updates per second: 0 ;               average distance vs reference: 5.71617e-05; min error : 0; max error : 0.000336332
State updates per second: 0 ;               average distance vs reference: 9.86522e-05; min error : 0; max error : 0.000500332
State updates per second: 0 ;               average distance vs reference: 0.000113735; min error : 0; max error : 0.000528291
State updates per second: 0 ;               average distance vs reference: 0.000134336; min error : 0; max error : 0.00066542
State updates per second: 0 ;               average distance vs reference: 0.000183676; min error : 0; max error : 0.000979989
State updates per second: 0 ;               average distance vs reference: 0.000222196; min error : 0; max error : 0.00108683
State updates per second: 0 ;               average distance vs reference: 0.000249959; min error : 0; max error : 0.0011195
State updates per second: 0 ;               average distance vs reference: 0.00029546; min error : 0; max error : 0.00144733
State updates per second: 124.59 ;               average distance vs reference: 0.00034518; min error : 0; max error : 0.0017722
State updates per second: 124.59 ;               average distance vs reference: 0.00039891; min error : 0; max error : 0.00206665
State updates per second: 124.59 ;               average distance vs reference: 0.000469698; min error : 0; max error : 0.00267574
State updates per second: 124.59 ;               average distance vs reference: 0.000533009; min error : 0; max error : 0.00310962
State updates per second: 124.59 ;               average distance vs reference: 0.00054639; min error : 0; max error : 0.00370819
State updates per second: 124.59 ;               average distance vs reference: 0.000597462; min error : 0; max error : 0.00494501
State updates per second: 124.59 ;               average distance vs reference: 0.000700545; min error : 0; max error : 0.00461178
State updates per second: 124.59 ;               average distance vs reference: 0.000790951; min error : 0; max error : 0.00608116
State updates per second: 124.59 ;               average distance vs reference: 0.000841109; min error : 0; max error : 0.00895351
State updates per second: 124.59 ;               average distance vs reference: 0.000908443; min error : 0; max error : 0.010299
State updates per second: 124.59 ;               average distance vs reference: 0.00103664; min error : 0; max error : 0.013408
State updates per second: 124.59 ;               average distance vs reference: 0.00110238; min error : 0; max error : 0.0211925
State updates per second: 124.59 ;               average distance vs reference: 0.00115024; min error : 0; max error : 0.0136486
State updates per second: 124.59 ;               average distance vs reference: 0.00131057; min error : 0; max error : 0.013997
State updates per second: 124.59 ;               average distance vs reference: 0.00147942; min error : 0; max error : 0.018734
State updates per second: 124.59 ;               average distance vs reference: 0.00155534; min error : 0; max error : 0.0192557
State updates per second: 124.59 ;               average distance vs reference: 0.00162146; min error : 0; max error : 0.02438
State updates per second: 124.59 ;               average distance vs reference: 0.00179331; min error : 9.53674e-07; max error : 0.0350508
State updates per second: 124.59 ;               average distance vs reference: 0.00192743; min error : 4.76837e-07; max error : 0.047497
^CState updates per second: 124.59 ;                 average distance vs reference: 0.00208586; min error : 4.76837e-07; max error : 0.0624085
State updates per second: 61.977 ;               average distance vs reference: 0.00231579; min error : 9.53674e-07; max error : 0.080265
```

MERCI