CONTACT NAME
Adam MacDonald: adam.macdonald@onezero.ca

Fluidix
Particle Simulation API

Adam MacDonald
OneZero Software, Fredericton, NB, Canada
adam.macdonald@onezero.ca
www.onezero.ca

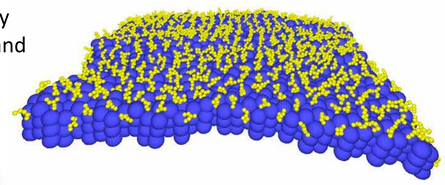
In association with:

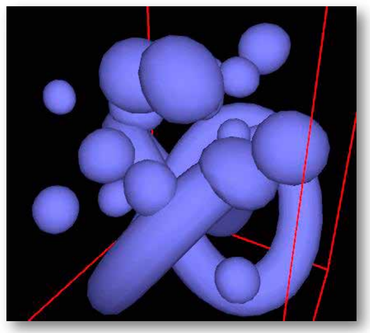
Envenio, Inc.
www.envenio.ca

University of New Brunswick
www.unb.ca

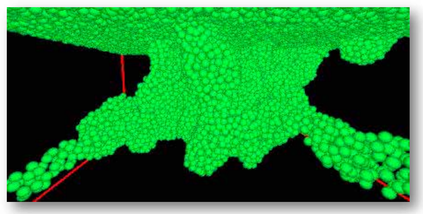
Abstract

Fluidix is a CUDA-powered particle simulation library which provides many of the low level parallel computation and memory-related tasks common to particle and mesh-based simulation methods.





Fluidix demonstrates that speed, flexibility, ease of use, and scalability are not mutually exclusive. It is implemented as a C++ template library, using functor classes to run callback-style device code at peak performance on the evolving CUDA architecture.



It provides an intuitive programming platform for modeling, executing, and visualizing 3D particle-based systems, enabling rapid development for small research teams with highly dynamic ideas.

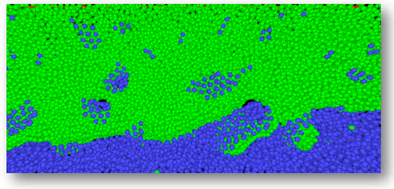
Simulation Modeling

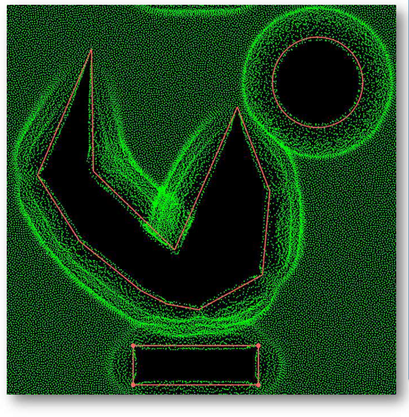
An **object** represents a simple data point in your model, based on the variables you need. An object can be many things, such as:

- Particle (position, velocity, acceleration)
- Vertex (3D mesh, model, or surface)
- Cell (centroid, data element of a grid)

An **interaction** represents a relationship between one or more sets of objects, based on the equations or computations you need. An interaction can operate on many things, such as:

- All objects in a set
- All nearby pairs of objects in a set
- All pairs predefined in a list
- All adjacent cells in a grid
- All vertices of a surface
- All points that have penetrated a 3D surface



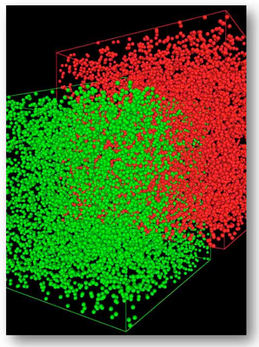


Objects can be created and modified on the fly, and interactions executed on the CPU or GPU in any order. You can export data and view results in 3D in real-time.

Fluidix functions as platform for new simulations, or can be integrated as a component into an existing simulation. It can run on any laptop, home PC, or multi-GPU workstation running Windows or Linux with a Fermi or later GPU.

Implementation

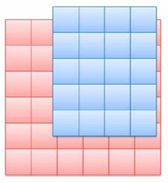
At the heart of Fluidix is a pair-particle search algorithm that finds and executes custom code in parallel on all pairs of particles within a specified cut-off distance in two sets. Particles are divided into a uniform grid, and only particles in nearby cells are included in an N^2 search for pairs.



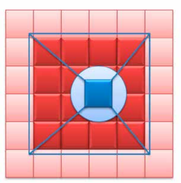
In Fluidix, many aspects of a simulation can change dynamically, such as the number of particles, the bounding box of a set, or the interaction cutoff range.

- Particle sorting and neighbor searching is done on the GPU, and is fast enough to be repeated each time-step.
- Sorting allows memory access to particle data to be highly coalesced.
- Each pair of particles is visited only once, using atomics for data updates. With Kepler, this is faster than double-counting without atomics.

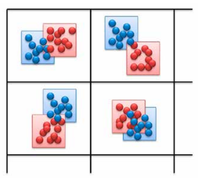
1. Organize each set of particles into its own uniform grid. Particle data is sorted using a Radix sort provided by the Thrust library.



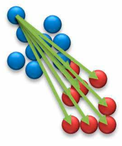
2. For each cell in grid A, identify a bounding box of neighboring cells in grid B.



3. Assign one block of threads to each pair of cells.

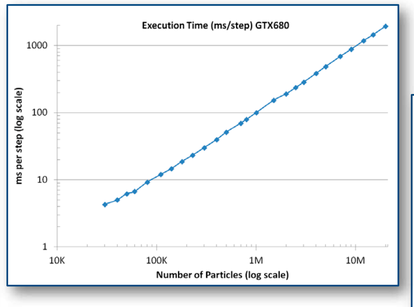


4. Assign one thread to each particle in cell 1, which contains a loop iteration for each particle in cell 2.

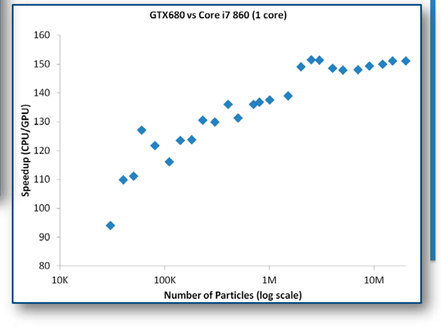


Performance

The pair-particle interaction represents the overwhelming majority of computation time in most particle-based fluid dynamics simulations. Benchmarks of this algorithm were performed using a single GTX680 and one CPU core of an Intel Core i7 860 at 2.8Ghz, using a simplified version of the example shown to the right.



Execution time of the algorithm is linearly related to both the number of particles and the density.

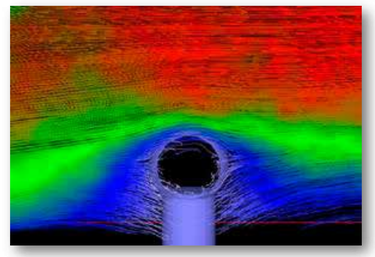


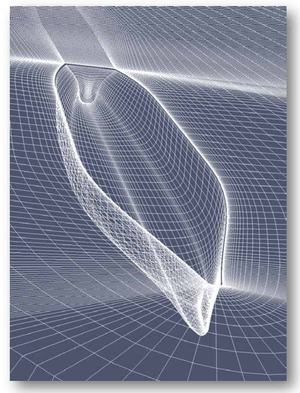
Execution time on the GPU is observed to be consistent for small and large systems, with speedup of up to 150.

Applications

Dissipative Particle Dynamics (DPD)

Fluidix was originally developed as a DPD simulation tool. With DPD interactions, Fluidix can model flowing water, dynamic interactive surfaces, polymers and other solid structures, and long-range electrostatics. Fluidix has been used to model bacterial growth in flowing water, stoppage of blood flow and a bypass, and antibiotics attacking a baterial surface.





Smoothed Particle Hydrodynamics (SPH)

SPH is a particle-based method for modeling large scale fluid flow, from water pouring out of a glass, to ocean waves crashing on a ship, to underwater explosions. SPH is especially versatile, and has been used in animation and film. Fluidix provides a platform for implementing highly customizable and complex interactions for SPH.

Ongoing research with Envenio Inc. involves developing a hybrid SPH/CFD model by combining Fluidix and EXN/Aero CFD software. CFD will model water beneath the surface, while SPH will handle the complex dynamics of waves on the surface.

Computational Fluid Dynamics (CFD)

In partnership with Envenio Inc., Fluidix has been integrated into the EXN/Aero CFD simulation software to provide a fast GPU solution to the over-set mesh problem. Fluidix is tasked to perform a per-cell interpolation on two dynamic, overlapping meshes. Cells are mapped to particles by centroid, nearby cells found by distance, and a volume intersection interaction is used. Ongoing research involves extending this solution to multiple overlapping meshes, and implementing a variable cut-off range to optimize the search.

Example

```
#include "fluidix.h"

struct MyParticle {
      xyz r, v, f; // position, velocity, force
      float a, b, c; // custom data elements
};

// linear particle repulsion
FUNC_PAIR(MyParticle, repulsion,
      xyz f = 100 * (1 - dr/2) * u;
      addVector(p1.f, f);
      addVector(p2.f, -f);
)

// gravity force
FUNC_EACH(MyParticle, gravity,
      addVector(p.f, make_xyz(0, -9.81, 0));
)

// boundary condition
FUNC_EACH(MyParticle, boundary,
      if (p.r.y < 0) {
            p.r.y = 0;
            p.v.y = fabsf(p.v.y);
      }
)

// basic Euler integration
FUNC_EACH(MyParticle, integrate,
      p.v += p.f * dt;
      p.r += p.v * dt;
      p.f = make_xyz(0, 0, 0);
)

// simulation
int main() {

      // initialize Fluidix
      Fluidix<MyParticle> *fx =
            new Fluidix<MyParticle>(0.01);

      // add particles
      int setA = fx->AddParticles(1000000);

      // randomize position
      fx->SetParticlesArea(setA, make_xyz(0, 0, 0),
                                 make_xyz(100, 100, 100));

      // simulation loop
      for (int i = 0; i < 10000; i++) {

            // begin time-step: sort particles
            fx->Start();

            fx->Interaction_Pair(setA, setA, 2.0, repulsion());
            fx->Interaction_Each(setA, gravity());
            fx->Interaction_Each(setA, boundary());
            fx->Interaction_Each(setA, integrate());

      }

      fx->Output("example");
      delete fx;
}
```