FluidSim
Parallel fluid particle simulation and visualization

by
Paul Bienkowski
2bienkow@informatik.uni-hamburg.de
1 – Introduction
1.1 Motivation
1.2 Particle Model
1.3 Force Model

2 – The Simulator
2.1 Parallelization
2.2 Synchronization
2.3 Data output

3 – The Visualizer
3.1 Basic Technology
3.2 Data Transfer

4 – Results
4.1 Did it work?
4.2 Live Demonstration
4.3 Performance
4.4 Difficulties
4.5 Problems

5 – Conclusion
Introduction
Motivation
Motivation
Idea
Simulate 2D-particles that repel each other
Motivation

Idea
Simulate 2D-particles that repel each other

Goal
Simulate an airplane wing and measure lift
Motivation

Idea
Simulate 2D-particles that repel each other

Goal
Simulate an airplane wing and measure lift

Technology
C++11, MPI, OpenMP, SFML
Each particle has 3 basic properties:

1. position
2. velocity
3. force
Each particle has 3 basic properties:

1. **position**
2. **velocity**
3. **force**

Only position and velocity need to be stored across iterations, force is recomputed every iteration.
Any two particles repel each other:

\[
force_i := \sum_j \text{force}(|p_i - p_j|) \cdot \text{norm}(p_i - p_j)
\]

The force on a particle affects its velocity:

\[
velocity_i := velocity_i + force_i \cdot dt
\]

The velocity of a particle affects its position:

\[
position_i := position_i + velocity_i \cdot dt
\]


\[
\text{force}(x) = \begin{cases} 
F \cdot \left(1 - \frac{x-T}{D}\right)^P & \text{for } 0 \leq x \leq D + T \\
0 & \text{otherwise}
\end{cases}
\]

where

- \(x\) is the distance between the particles.
- \(F\) is the force strength factor
- \(D\) is the influence distance
- \(T\) is the distance threshold (particle radius)
- \(P\) is the force power
**Force Model**

\[
\text{force}(x) = \begin{cases} 
F \cdot \left(1 - \frac{x-T}{D}\right)^P & \text{for } 0 \leq x \leq D + T \\
0 & \text{otherwise}
\end{cases}
\]

**Changing the Force Power (P)**

- \(F = 1\)
- \(D = 1\)
- \(T = 0\)
\[ \text{force}(x) = \begin{cases} 
F \cdot \left(1 - \frac{x - T}{D}\right)^P & \text{for } 0 \leq x \leq D + T \\
0 & \text{otherwise} 
\end{cases} \]

Changing the Distance Threshold \((T)\)

\[
\begin{align*}
F &= 1 \\
D &= 1 \\
P &= 10
\end{align*}
\]
**Force Model**

\[
\text{force}(x) = \begin{cases} 
F \cdot \left(1 - \frac{x-T}{D}\right)^P & \text{for } 0 \leq x \leq D + T \\
0 & \text{otherwise}
\end{cases}
\]

I found these values work for the wing simulation:

\[
D = 0.001 \\
T = 0.06 \\
P = 1 \\
F = 20
\]
A mesh is a simple polygon, each segment (line) is checked for collision with every particle.
A mesh is a simple polygon, each segment (line) is checked for collision with every particle.

Simple linear algebra calculations are made for reflecting particles off mesh segments.
A mesh is a simple polygon, each segment (line) is checked for collision with every particle.

Simple linear algebra calculations are made for reflecting particles off mesh segments.

This creates a force on the mesh.
The Simulator
Parallelization

- each particle has to be updated (simple for loop)
  → threading trivial with OpenMP
- particles can be distributed across multiple processes
  → Domain Grid
Parallelization

- each particle has to be updated (simple for loop)
  → threading trivial with OpenMP
- particles can be distributed across multiple processes
  → Domain Grid

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4</td>
<td>P5</td>
<td>P6</td>
<td>P7</td>
</tr>
<tr>
<td>P8</td>
<td>P9</td>
<td>P10</td>
<td>P11</td>
</tr>
</tbody>
</table>
Synchronization

(a) Checkerboard

(b) Stripes

Modes of Domain coloring

<table>
<thead>
<tr>
<th>Mode</th>
<th>Sending</th>
<th>Receiving</th>
<th>Directions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Checkerboard</td>
<td>black</td>
<td>white</td>
<td>N, E, S, W</td>
</tr>
<tr>
<td>2. Checkerboard</td>
<td>white</td>
<td>black</td>
<td>N, E, S, W</td>
</tr>
<tr>
<td>3. Stripes</td>
<td>black</td>
<td>white</td>
<td>NE, SE, SW, NW</td>
</tr>
<tr>
<td>4. Stripes</td>
<td>white</td>
<td>black</td>
<td>NE, SE, SW, NW</td>
</tr>
</tbody>
</table>
### Data output

<table>
<thead>
<tr>
<th>00000000</th>
<th>0020 0000</th>
<th>0001 0000</th>
<th>0004 0000</th>
<th>2710 0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
<td>1387 0000</td>
<td>0000 0000</td>
<td>0000 0000</td>
<td>1389 0000</td>
</tr>
<tr>
<td>00000020</td>
<td>8a72 d187</td>
<td>c4bb 3fa1</td>
<td>73d1 4e75</td>
<td>d95f 3fbe</td>
</tr>
<tr>
<td>00000030</td>
<td>d3f4 2c9a</td>
<td>cf42 3fbc</td>
<td>b890 9e75</td>
<td>5a61 3fc1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00013a7</td>
<td>9aa6 af8e</td>
<td>c353 3fc7</td>
<td>1df2 1539</td>
<td>3ec1 3fc3</td>
</tr>
<tr>
<td>00013b7</td>
<td>c065 7825</td>
<td>f853 3fb3</td>
<td>717a c31f</td>
<td>1f55 3fc3</td>
</tr>
</tbody>
</table>

- **Header length:** 32 bytes
- **Iteration number:** 1
- **Number of processes:** 4
- **Particle count:** 10000
- **Particle count by process:** 4999 / 0 / 5001 / 0
- **Particle positions of P1** ($x_1, y_1, x_2, y_2, ...$)
- **Particle velocities of P1**
Basic Technology

- SFML for window, input, rendering (OpenGL inside)
- Load Iterations into memory
- Play/Pause/Live
- Different display modes (coloring of particles)
How is data transferred from Simulator to Visualizer?
How is data transferred from Simulator to Visualizer?

- Socket/Network/MPI?

SSHFS, status file contains metadata: number of iterations, grid size. Visualizer reads status in regular intervals.
Data Transfer

How is data transferred from Simulator to Visualizer?

- Socket/Network/MPI? → too complicated
How is data transferred from Simulator to Visualizer?

- Socket/Network/MPI? → too complicated
- SSHFS 😊
Data Transfer

How is data transferred from Simulator to Visualizer?

- Socket/Network/MPI? → too complicated
- SSHFS 😊
- status file contains metadata
  - number of iterations
  - grid size
- visualizer reads status in regular intervals
Results
Did it work?
Did it work?
Performance

All measurements were made with IO disabled, no particle data was recorded.
Performance

All measurements were made with IO disabled, no particle data was recorded.
All measurements were made with IO disabled, no particle data was recorded.
Difficulties

- MPI file input/output was hard to get working, miscalculated seek offsets etc.

Results
Difficulties

- MPI file input/output was hard to get working, miscalculated seek offsets etc.
- lots of segmentation faults and uninitialized values 😊

Cannot insert: Count 0xdeadbeed >= Size 0xdeadbeed.
Aborted.

Segmentation fault.
(gdb) frame 3
(gdb) print buf
$1 = 0xdeadbeedddeadbeed;
Difficulties

- MPI file input/output was hard to get working, miscalculated seek offsets etc.
- lots of segmentation faults and uninitialized values 😊

```plaintext
Cannot insert: Count 0xdeadbeef >= Size 0xdeadbeef.
Aborted.

Segmentation fault.
(gdb) frame 3
(gdb) print buf
$1 = 0xdeadbeedddeafbeed;
```

- 2D collisions are not *that* trivial
Difficulties

- MPI file input/output was hard to get working, miscalculated seek offsets etc.
- lots of segmentation faults and uninitialized values 😊
  Cannot insert: Count 0xdeafbeed >= Size 0xdeafbeed.
  Aborted.

  Segmentation fault.
  (gdb) frame 3
  (gdb) print buf
  $1 = 0xdeafbeeddeafbeed;

- 2D collisions are not *that* trivial
- the model (uplift) did not work out until I implemented surface damping
Problems

- software not optimized for RAM usage
  → can’t run more than 6 processes locally 😞
Problems

- software not optimized for RAM usage
  → can’t run more than 6 processes locally 😞
- memcpy is slow
Problems

- software not optimized for RAM usage → can’t run more than 6 processes locally 😞
- memcpy is slow
- 16 send/receive operations on a 12-node cluster, probably room for improvement,
Problems

- software not optimized for RAM usage
  → can’t run more than 6 processes locally 😞
- memcpy is slow
- 16 send/receive operations on a 12-node cluster, probably room for improvement,
Problems

- software not optimized for RAM usage
  → can’t run more than 6 processes locally 😞
- memcpy is slow
- 16 send/receive operations on a 12-node cluster, probably room for improvement, *however* this method scales to every cluster size
Conclusion
Yes, it works!
Even in real-time!
Even though $O(n^2 + nm)$ with $n \in O(10000)$
It looks kind of fancy...
I learned a lot.
Further works

- Load Balancer, based on number of particles in rows/columns
Further works

- Load Balancer, based on number of particles in rows/columns
- SIMD
Further works

- Load Balancer, based on number of particles in rows/columns
- SIMD
- “Ball” particle model (elastic collision of circle shapes)
Further works

- Load Balancer, based on number of particles in rows/columns
- SIMD
- “Ball” particle model (elastic collision of circle shapes)
- different World Scenarios / presets (gravity, water, ...)

Conclusion
Further works

- Load Balancer, based on number of particles in rows/columns
- SIMD
- “Ball” particle model (elastic collision of circle shapes)
- different World Scenarios / presets (gravity, water, ...)
- animation export
Thank you for your attention!

Questions?