Concordia University - Fall 2016
COMP 477 - Animation for Computer Games
Andres Gonzalez-27753675, Dexter Ho-Yiu Kwok-27709110, Jonathan De Marco-26941249
David Katchouni-26461891

Fluid Simulation using SPH

DESCRIPTION

The goal of this project was to create an application which simulates the movement and interaction of objects with fluids. We used a computational method called Smoothed-Particle Hydrodynamics (SPH) to model our particle interaction. Our application was developed with C++ and the modern OpenGL library (4.0+) with GLSL shader language. As for the computational method for the simulation of fluids, we did not use any external libraries to build our particle system.

Our application features a GUI menu from which the user has four options: create a new simulation, play an existing simulation, continue a simulation, or edit a scene. If the user decides to create a new simulation, he will be prompted to load a parameters file which specifies the particle systems parameters to use for the simulation. Afterwards, he will be prompted with the amount of particles he wants to simulate and the file to save the simulation to. The user can also load an existing simulation, continue calculating an existing animation, or edit a scene.

Once an animation is loaded into the scene, it can be played and paused by using the corresponding buttons or by pressing p. The user can move the camera in the scene with the mouse and keyboard (WASD).

MOTIVATION

The motivations for our project were to

- Understand how fluid dynamics work.
- Understand the challenges of fluid simulation.
- Implementation of SPH in an application and observe fluid characteristics.

- Create realistic-looking environments.

Design / Algorithms

Neighbor Search:

We implemented a uniform 3D grid data structure to do our neighbor search. Each cell in our grid stores a list of indices that reference the particles who are in that cell. These indices represent the index of the particle in our particle system particle list. We also set the grid cell size to the neighbor search radius. This allows us to significantly speed up our neighbor search. A particle simply searches his 26 adjacent cells and his own..

Memory Speed:

During the development of the project we ran into many problems with memory. We were storing the particle positions per frame in a 2-dimensional array. When we played or calculated an animation with a large amount of particles, the application would use a very large amount of memory since this data structure would grow so big. We did two things to tackle this problem:

1. **Calculations:** When calculating our particle positions using SPH, we decided every time a frame is calculated to write it to a file.

2. **Play**: For playing our animation, we decided to load our animation file in chunks. The algorithm we implemented runs in parallel and works as follows:

We have 2 frame buffers; a front buffer and a back buffer. Initially the front buffer loads a fixed amount of frames (e.g. 60) from the file. While we render the particles using the front buffer, we load the next 60 frames in parallel into the back buffer. Once the front buffer reaches its final frame, it swaps buffer with the back buffer, and the process repeats.

SPH:

The computational method for the simulation of fluids was determined by Smooth Particle Hydrodynamics. The density gradient, pressure gradient, laplacian velocity and the color field (surface tension) were calculated by using smoothing kernels. For each individual property, it had its specific kernel. As presented in class, the density used the *poly5* kernel, the pressure used the *spiky* kernel, and the laplacian velocity used the *viscosity* kernel. The implementation of the calculations was very simple, however, many operations were done twice. The assignment of indices to each particle allowed us to sum the values of properties simultaneously for pairs of

particles due to particle-particle interaction. As a result, calculations were cut down by half as each particle object had a variable for each property which was required for the calculation of the forces. The algorithm looped through each particle's neighbors and calculated its property and its neighbor's property. As for the overall calculation, the algorithm is separated into five different sections which consists of the calculation of the neighbors, the densities and pressure with respect to the neighbors, the internal forces and the collisions. These sections were separated in order to allow a simple integration of OpenMP (Parallel Programming).

Rigid Bodies & Collisions:

Collisions with rigid bodies were handled by predefining all intersections between the mesh triangles and the grid cell structure detailed above. A pointer to the intersecting rigid body, as well as the index of the triangle in question is stored in each colliding cell so as to speed up calculations. After the SPH calculations and Euler time integration phases, the collision algorithm searches neighboring cells for collisions, and does the appropriate checks (whether the distance from center of particle to plane is less than the radius, etc). It then proceeds to move particles back in the direction whence they came if they do collide, and reflects their current velocity about the normal of said plane. Although 5 distinct collision cases are taken into account based on position relative to the plane and velocity, it has proven challenging to eliminate all particle leaks from the rigid body containers, especially with greater numbers of particles.

Difficulties

One of the main difficulties involved in our project were figuring out how to implement SPH and tweaking the parameters to get a nice simulation. We also had difficulty optimizing the code. We did the following to optimize our SPH code:

- Parallelize SPH loops using OpenMP
- Applying symmetry to particle-particle interactions (e.g. pressure, density, gradient, …)

In the end, between the first iteration of our program and the current iteration, we had a speedup factor of 12.

Furthermore, we had difficulties dealing with memory and rigid body collisions. There were certain collisions of particles which were not being triggered. We tried to solve this issue by changing the result of one of the conditions. We backtracked the particle's to the point where it is about to collide with the surface and reflect its velocity. The difficulty lied in the debugging aspect of the code. It was very hard to pinpoint where the error was located.

Work-Load Distribution

Andres Gonzalez - Collisions, Simulator Engine, Ray tracing, and rigid body structures
Dexter Ho-Yiu Kwok - SPH, Parallel Programming, Testing and Simulation Parameters
Jonathan De Marco - Spatial Data Structure, Parallel Programming, Serialization, File I/O
David Katchouni - Scene View/Editor Mode, Camera Interaction, Menu and Pause/Play interaction

Compilation

The project should be compiled with Visual Studio 2013 or 2015. Depending on the version of Visual Studio you might use to compile the source code, different lib files will be needed. The two libraries affected by this are GLFW and Freetype. For GLFW, you must change the include directory to include $(SolutionDir)\Resources\GLFW\lib-vc2015 or $(SolutionDir)\Resources\GLFW\lib-vc2013, depending on whether you are using Visual Studio 2013 or 2015. For Freetype, the one included in the project is for Visual Studio 2013. If you are using Visual Studio 2015, we have include a lib file in Resources/Freetype/2015 where you can find the lib file Freetype needs for Visual Studio 2015. Replace the Freetype lib file in both the Release folder and the one found in $(SolutionDir)\Resources\Freetype\lib. Make sure you are running Visual Studio in Release 32bit (x86).

External Resources

---

[1] Rice, E. OpenGL GPU Features and SPH Fluid - Computer Animation and Visual Effects Msc, Bournemouth University, Aug. 22, 2016

[2] Siggraph 2013 Course: Physically Based Shading in Theory and Practice, http://blog.selfshadow.com/publications/s2013-shading-course/

[3] Green, S et al., Fluid Rendering Alice, http://developer.download.nvidia.com/tools/docs/Fluid_Rendering_Alice.pdf

[4] Harris, M., CUDA Fluid Simulation in NVIDIA PhysX,
http://sa08.idav.ucdavis.edu/CUDA_physx_fluids.Harris.pdf

[5] Muller, M., Particle-Based Fluid Simulation for Interactive Applications
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.2.7720&rep=rep1&type=pdf

[6] Kelager, M., Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics
http://image.diku.dk/projects/media/kelager.06.pdf