This exercise is optional; do it whenever you like.

In this exercise, you will implement a real-time smoothed particle hydrodynamics fluid simulation on the GPU (the graphics card).

**GPU computing.** Since hydrodynamics simulations need to perform large numbers of similar computations (either for the particles in particle-based methods, or for the grid cells in grid-based methods), they hardly achieve interactive framerates when implemented on the CPU. Modern graphics hardware, however, is capable of computing the movement of many particles in parallel, and can therefore simulate fluid dynamics in real time.

**CUDA.** We will use nvidia’s **CUDA** [http://www.nvidia.com/cuda](http://www.nvidia.com/cuda) for programming the GPU. CUDA programs can be written in a C/C++ dialect, so that you can use the computing power of your GPU without learning a new programming language. You do, however, need a recent nvidia GPU in order to be able to use CUDA. The CIP pool computers are all CUDA capable.

**OpenCL.** If you want to solve this exercise on a computer that does not support CUDA, you may have a look at OpenCL instead [http://www.khronos.org/opencl/](http://www.khronos.org/opencl/). While the basic concepts of CUDA and OpenCL are similar, the CUDA programming environment is somewhat easier to use in practice, but you are free to implement this exercise using OpenCL if you prefer.

**Kernels.** CUDA programs consist of so-called kernels. A kernel is a small program that runs on many GPU processors in parallel, much like a fragment shader. In CUDA, kernels are defined like C functions, but the function declaration is prefixed with the keyword `__global__`. You can mix CUDA kernels and ordinary C functions in a single file. By convention, such files end in `.cu`, and they are compiled with the nvidia compiler `nvcc` which internally uses `gcc` to compile the C part of the source file.

**Executing kernels.** A kernel function is called like a C function, except that the function name is followed by additional parameters in triple angle brackets (this is already done for you in the code framework). These additional parameters specify how often the kernel is to be executed. Namely, the first parameter determines how many “blocks” are executed, and the second parameter controls how many threads each block contains. For this exercise, you do not need to care about how threads are organized into blocks. You only need to know that each thread has a unique index `i` (see the code framework) which is defined by the index of its block and the index of the thread within the block.

**GPU memory.** If you want to compute an operation on an array of numbers in parallel using a kernel, you can pass a pointer to this array to the kernel. Each thread will compute its unique index and perform the operation on the corresponding element of the array. Of course, you can pass pointers to multiple
arrays, some of which are read from, while others are written to. Note that all pointers that you pass to kernels refer to GPU memory. There are special functions that allocate and deallocate GPU memory (cudaMalloc and the like), and functions to copy data from CPU to GPU memory (cudaMemcpy). But since the code framework already handles memory allocation, you do not need to care about this for now.

AntTweakBar. The code framework also makes use of the AntTweakBar library for displaying and modifying the simulation parameters interactively from within the OpenGL window. A 64-bit Linux binary of the library is provided in the code framework. You can also download the code from [http://www.antisphere.com/Wiki/tools:anttweakbar:download](http://www.antisphere.com/Wiki/tools:anttweakbar:download) and compile it yourself. You may have to #include cstdlib, cstdio and cstring in src/TwPrecomp.h before you execute make in the src directory.

Smoothed particle hydrodynamics. Originally developed for astrophysical simulations, this particle-based simulation approach has been adapted for computer graphics because it is fast, easy to implement, and easy to render. The basic idea is that each particle represents a fluid blob of finite size with a density that gradually lowers towards the boundary of the blob. This allows to represent a continuous fluid by a small but discrete set of particles. For details, I suggest that you have a look at the following paper, which is quite readable despite the large amount of math involved: Müller et al., “Particle-based fluid simulation for interactive applications”, 2003. [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.7720](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.7720)

12.1 Collision with a plane (25 points)
The smoothed particle hydrodynamics simulation is based on particles that move according to Newton’s equations of motion. Reflection planes have been defined to make sure that each individual particle stays confined within the screen area. Implement the kernel collide_plane in main.cu so that the particle is reflected by the respective plane. Use continuous collision detection, that is, analytically compute where the particle will be (and what its velocity will be) after the collision, assuming the particle moves linearly with constant velocity during each time step. Multiply the normal component of the velocity by $1 - \text{damping}$ to make the collisions less “bouncy”.

12.2 Density computation (25 points)
You will notice that the particles stay within the screen area, but they still do not interact with each other. Before any interaction forces can be computed, we have to compute an approximate mass density of the fluid at the position of each particle. The mass density can be computed by convolving the particles’ point mass distribution with a kernel

$$W_{\text{density}}(\vec{r}, h) = \frac{315}{64 \pi^9} \left( h^2 - |\vec{r}|^2 \right)^3.$$  

Here, $h$ is the kernel size, and $W_{\text{density}}(\vec{r}, h) = 0$ for $|\vec{r}| > h$. Implement compute_density in main.cu.

12.3 Pressure force (25 points)
Now the approximated mass densities at the particle positions are stored in the density array and can be used to compute the interaction force due to pressure between the particles. The pressure force for the particle with index $i$ can be computed as

$$F_{\text{pressure}} = \sum_{j \neq i} \frac{45 k m_i}{\pi h^6} \frac{\rho_i + \rho_j - 2 \rho_0}{2 \rho_j} \frac{(|\vec{r}_i - \vec{r}_j| - h)^2}{|\vec{r}_i - \vec{r}_j|} \cdot (\vec{r}_i - \vec{r}_j),$$

where $k$ describes the strength of the interaction, $m_i$ is the mass of the particle with index $i$, $h$ is again the kernel size, $\rho_i$ and $\rho_j$ are the mass densities at the position of the corresponding particles, $\rho_0$ is a density offset for numerical stability, and $\vec{r}_i$ and $\vec{r}_j$ are the positions of the particles $i$ and $j$. For $|\vec{r}_i - \vec{r}_j| > h$, the force is again zero. Implement add_pressure_force in main.cu.
12.4 **Viscosity force (25 points)**

The particles now correctly push each other apart, but a real fluid also has internal “friction” forces that make adjacent particles perform similar movements. These forces can be modeled as

\[
\vec{F}_{\text{viscosity}} = \sum_{j \neq i} \frac{45 \mu m_i}{\pi h^6} \frac{h - |\vec{r}_i - \vec{r}_j|}{\rho_j} (\vec{v}_j - \vec{v}_i)
\]

where \(\mu\) describes the strength of the interaction, and \(\vec{v}_i\) and \(\vec{v}_j\) are the velocities of the particles \(i\) and \(j\). For \(|\vec{r}_i - \vec{r}_j| > h\), the force is again zero. Implement `add_viscosity_force` in `main.cu`.

12.5 **Surface tension (25 points)**

Finally, fluid particles also attract each other. This has no effect within the fluid, where every particle is uniformly surrounded by other particles, but it affects particles at the surface of the fluid. This force can be modeled as

\[
\vec{F}_{\text{surface}} = \sum_{j \neq i} \frac{945 \sigma m_i}{32 \pi h^9} \frac{h^2 - |\vec{r}_i - \vec{r}_j|^2}{\rho_j} \left(3h^2 - 7|\vec{r}_i - \vec{r}_j|^2\right) \frac{\vec{n}}{|\vec{n}|}
\]

where \(\sigma\) describes the strength of the interaction, and \(\vec{n}\) is defined as

\[
\vec{n} = \sum_{j \neq i} \frac{945 m_i}{32 \pi h^9} \frac{h^2 - |\vec{r}_i - \vec{r}_j|^2}{\rho_j} (\vec{r}_i - \vec{r}_j)
\]

For \(|\vec{r}_i - \vec{r}_j| > h\), or when \(|\vec{n}| < \text{surface\_threshold}\), the force is zero. Implement `add_surface_force` in `main.cu`.

12.6 **Collision with a sphere (25 points)**

You will probably want to be able to interact with the fluid to see how it reacts. You can move a sphere around with the mouse; in order to compute collisions of particles with the sphere, implement `collide_sphere` in `main.cu`. Proceed as you did for collisions with a plane: first, compute where the particle has hit the surface, then mirror the particle position at this point, reflect the normal component of the velocity and apply the specified amount of damping.

12.7 **Explore the parameter space (25 points)**

You have noticed that there are many parameters that can be tweaked in order to achieve different results. Try to simulate water, honey, mud, smoke, a lava lamp, play dough, foam... What can be done with your algorithm? What can’t? Why? How could you extend the algorithm to make it possible?

http://graphics.tu-bs.de/teaching/lectures/ws1314/pbm/