UCLouvain

École Polytechnique de Louvain

LMECA2300 : Advanced Numerical Methods

Smoothed Particles Hydrodynamics

April 2020

Jimmy Fraiture	1711 16 00
Michel Henry	6396 15 00
Nicolas Bioul Kamborian	8421 14 00

Group 6

2019 - 2020

1 Introduction

This project focused on the development of a great animation using shaders. The animation describes the movement of a fluid driven by its boundary. The report provides the main features introduced in the code and how the implementation of shaders has been done.

2 Physical Model

The fluid motion will be described using a set of particle representing the material particles. The approximation method is based on the smoothed particle hydrodynamics method (SPH). These particles has to respect the conservation law. Considering a circle in which the boundary rotates at a given velocity, the boundary motion will drive the fluid into a circular movement. The situation is described in figure (1).

The fluid flow is supposed to be incompressible, Newtonian and inviscid. Therefore, the conservation laws can be reduced to :

$$\begin{cases} \dfrac{D\rho}{Dt} = 0, \ \nabla \cdot \mathbf{u} = \mathbf{0} \\ \dfrac{D\mathbf{u}}{Dt} = -\dfrac{1}{\rho}\nabla P, \end{cases}$$

where ρ is the density, $\mathbf{u}$ the velocity vector, ∇P the pressure gradient. The pressure is described using the Tait's equation,

$$P = B\left(\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right).$$

The constant ρ_0 is the fluid density at rest, γ is a reference value oftenly chosen to be 7 and B is a term associated to the fluctuations of fluid density [2].

The boundary condition appears as a Dirichlet condition on the velocity field. The prescribed velocity is denoted by $\mathbf{u}_\Gamma$ and the boundary is denoted by Γ.

$$\mathbf{u} = \mathbf{u}_\Gamma, \qquad \text{on} \quad \Gamma$$

3 SPH Model

In order to simulate accurately the physics, some features may have the interest to be specified in this report.
First of all, artificial viscosity is introduced in order to avoid particles superposition. This has been done using XSPH with a parameter sets to 0.5.
Secondly, the density is filtered at each time step using CSPM to restore the zero-th consistency condition. Its purpose is to ensure the density will not change once the stationary regime is reached.
Finally, to impose the Dirichlet condition the update on the boundary has to been adapted. In this experiment, the imposed velocity is always tangent to the boundary. Therefore once a particle is detected to be outside of the domain, the reflective boundary condition have to impose the velocity tangent to the edge of this particle and adapt its center. The update velocity therefore becomes :

$$u_t = u_\Gamma, \qquad u_n = 0$$

It come from the fact that the kinetic energy restitution coefficient CR is imposed to 0 to vanish the normal component. The resulting time integration algorithm is described in algorithm 1.

Algorithm 1: Moving Circle Simulation

```
initialization;
while t < tEnd do
    update cells;
    update neighbors;
    compute density;
    filter density using CSPM;
    compute pressure using Tait's equation;
    update velocity;
    correct velocity using XSPH;
    animation;
    t += dt;
end
```

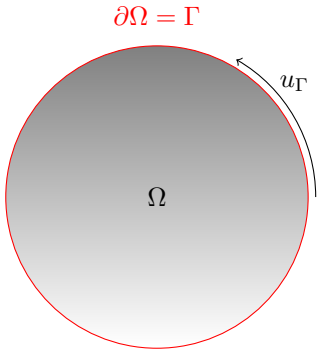

Figure 1: Moving boundary.

The simulation works in two steps, first the particles are initialised, since their repartition into the circle is not uniform, the density fields is at first not constant. It will leads to a various pressure field. This variation will lead to a pressure force applied to the particles. The particles in which the pressure is higher will move towards the particles where the pressure is lower leading to a smoothed pressure and density fields. Then the second phase begins once particles touch the boundary, their velocity is adapted and the motion begins.

4 Shaders

The shaders are the script used to display the elements of the screen. For this project the particle's shaders has been modified. Their organisation is described in figure 2.

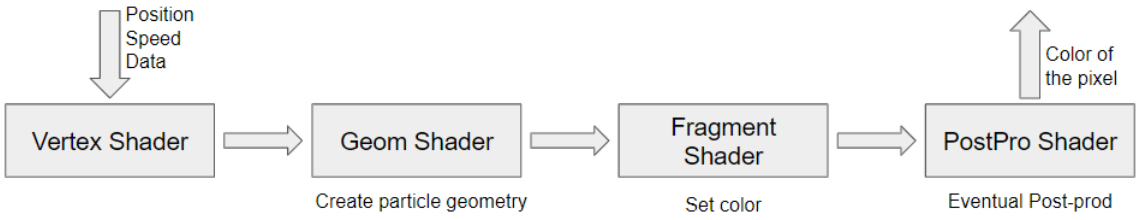

Figure 2: Shader files organisation.

Since the majority of the modification has been done on the color of the particles, the fragment shader is the shader in which we apply our modification.
To improve the visual quality of the simulation, some shaders and other improvement have been added. All of them are available through a side menu on the simulation window. On this menu, the user can choose to display different versions of the animation.

4.1 Continuous fields

The physical fields, i.e. pressure, density and velocity norm, are plotted in continuous fields. This has been done using shaders. We will describe the methodology for the pressure but it is the same for the other fields. First, the rendered size of each particle is increased by a factor 1.8 to remove empty spaces between particles (minimum size should be $\sqrt{2}$ if particles are touching each other.
Then at each point of the domain we need to have the pressure. For that we will use OpenGL's alpha blending to have the information of all the particles around.
The purpose is to get a linear interpolation of pressure at each point based on the surrounding particles. Using the alpha blending formula [1]:

$$(R,G,B) = \frac{(R_s,G_s,B_s)A_s + (R_d,G_d,B_d)A_d(1-A_s)}{A_s + A_d(1-A_s)} \qquad s = \text{source},\ \ d = \text{destination}$$

and by choosing the colors as :

(R,G,B,A) = (Distance to center x Pressure of the particle, 1, Distance to center, 0.2)

it is possible to interpolate the pressure of a point. Indeed in OpenGL, when a color is put above another with an alpha, OpenGL need to blends these colors using the previous equation. Based on this equation we can see that the ratio between the blue and red color is the same and will act as an interpolation of the pressure at one point.
The post production shader will then apply the following formula on each pixel with a green component equal to 1 (each pixel is influenced by particles).

Algorithm 2: Color post production

```
current_color = current_pixel.color.rgb;
intensity = current_color.r/current_color.b;
outColor = colormap(intensity)
```

$$\begin{cases} \text{Red} & = 1.5 - 4|\text{intensity} - 0.75| \\ \text{Green} & = 1.5 - 4|\text{intensity} - 0.5| \\ \text{Blue} & = 1.5 - 4|\text{intensity} - 0.25| \\ \text{Alpha} & = 1 \end{cases}$$

Figure 3: Jet colormap

The alpha is set sufficiently low (0.2) such that even with 6 particles the information of them is never loss (6 particles being the maximum we can get).
An important thing to note is that the post production shader will work on the RBG color and not the RGBA such that the alpha blending is fully applied before the application of the colormap.

Note : *the plotted information is always the relative field at a point according to the rest of the domain. So if a region is red it doesn't means that the pressure is "really high" but it means that it is the highest pressure of the field (same for density and velocity)*

With this method, the continuous fields of different attribute of the domain can be visualised during the simulation. Since all the computation are GPU, it doesn't slow down the simulation.
The result for the velocity field can be found on Figure 5 to illustrate the method.

4.2 Light

This mode simulate a light source somewhere in the field. It will give a light diffusion effect and create a shadow generated by the particles. To produce this effect, 2 steps are one.
First, we create a light particle with a special marker so that the shaders recognize it. It will render this light particle as a fade in the alpha component over the whole domain : $\text{outColor.a} = 0.6(\text{distToLight}) + 0.4$ for each pixel of the domain. This choice of weighting is arbitrarily chosen for a nice rendering.
Then, over this light, the real particles are plotted and for each of them 3 triangles are plotted, to represent the shadow polytope, building the shadow generated by the particle and computed by projecting the light on the circle of the particle.

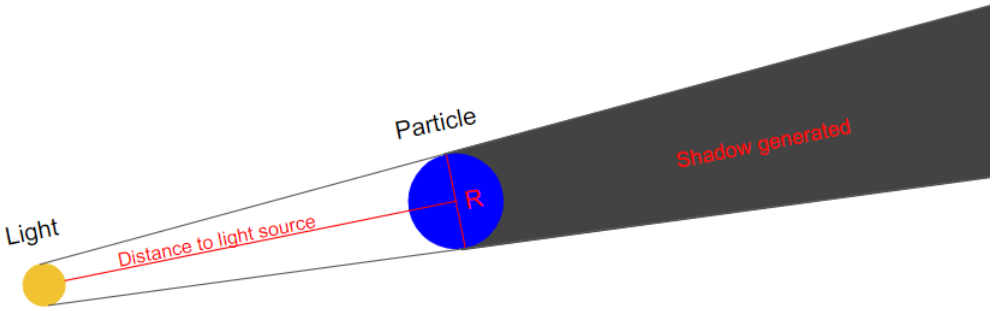

Figure 4: Shadow computing

Here is the result for a low number of particles

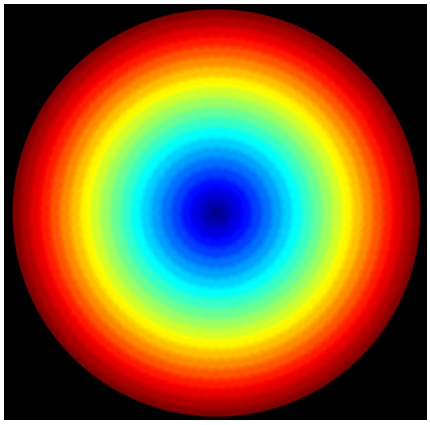

Figure 5: Norm of the velocity

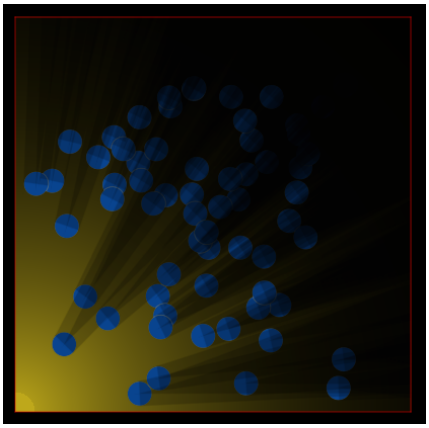

Figure 6: Example of light shader

Note : *this could be improved if we had access to the triangle shaders to be able to detect if a particle is touched by light or not and so modify the particle's color according to that. Otherwise the computationnal cost would be too high.*

References

[1] Apoorva Joshi. *Alpha compositing, OpenGL blending and premultiplied alpha*. 2019. URL: `https://apoorvaj.io/alpha-compositing-opengl-blending-and-premultiplied-alpha/`.

[2] *Smoothed Particle Hydrodynamics*. Springer International Publishing, 2019.