AQUAgpusph, a free 3D SPH solver accelerated with OpenCL

J.L. Cercos-Pita, A. Souto-Iglesias, L.M. Gonzalez, F. Macià
Model Basin Research Group (CEHINA V), Naval Architecture Dept.(ETSIN), Technical University of Madrid (UPM), 28040 Madrid, Spain.
jl.cercos@upm.es

Abstract—In this paper AQUAgpusph, a new free SPH software licensed under GPLv3 and accelerated using OpenCL, will be described. Its main differences with respect to other GPU based SPH implementations will be discussed, focusing first on the fact that is accelerated with OpenCL, second on the wide range of solid boundary condition enforcing methods have been implemented (including boundary integrals) and finally on the possibilities that offers its customization via Python scripting.

I. Motivation

In recent years, numerical computer simulations have suffered an unexpected paradigm change with the introduction of the GPGPU technique, that allows for the use of graphic oriented processing units for more general purposes. These devices have the same computational capabilities than a middle size CPU based cluster for a fraction of the cost, feature that quickly attracted the attention of the researchers in almost every area of science and engineering.

Traditionally, these new paradigm based applications have been developed either in assembly language, or using a graphic implementation specific language, such as GLSL, HLSL or Cg, but with the increasing interest in this technique some manufacturers and researchers have developed high level tools that do more accessible GPU oriented programming to the general public, at which most famous is probably the CUDA developing language [1], but several many others has been presented too, e.g. BrookGPU, CTM, CBEA or Sh.

Regarding the CPUs, a technical wall was reached few years ago due to the impossibility of increasing the processor’s clock-rate. The manufacturers dealt with this situation focusing their developments on the multi-core chips with a clock-rate stabilized around 3 GHz. In order to use this new feature, specific new libraries were mandatory, such that most operating systems made their own implementations, although actually the open standard OpenMP [2] is usually applied for all operating systems and architectures.

Therefore a lot of different frameworks exist to develop multi-core parallelized codes, each of them designed for different operating systems or architectures. In order to unify the software development for all these different multi-core alternatives, an open standard has been proposed, the Open Computing Language (OpenCL) [3]. Actually, all major vendors (Nvidia, AMD, IBM, Intel, etc.) have adopted this standard, allowing a unique implementation to be used for massive parallel computations in a wide variety of architectures. The OpenCL features will be discussed later.

Particularly for the Smoothed Particles Hydrodynamics (SPH) method, several implementations have proven the capabilities of graphic oriented devices to perform massive computations using CUDA language [4]–[7].

AQUAgpusph is designed to solve approximately the incompressible Navier-Stokes equations, using for that the weakly-compressible SPH (WCSPH) technique.

II. Governing equations

AQUAgpusph is designed to solve approximately the incompressible Navier-Stokes equations, using for that the weakly-compressible SPH (WCSPH) technique.
The compressible Navier-Stokes equations for a barotropic fluid in Lagrangian formalism are:

\[
\begin{align*}
\frac{Dp}{Dt} &= -\rho \operatorname{div}(\mathbf{u}) \\
\frac{Du}{Dt} &= g + \frac{\operatorname{div}(\mathbf{T})}{\rho} \\
p &= p(\rho)
\end{align*}
\]  

(1) 

(2) 

(3)

Here, \( \rho \) is the fluid density, \( p \) is the pressure and \( g \) is a generic external volumetric force field.

The flow velocity, \( \mathbf{u} \), is defined as the material derivative of a fluid particle position \( \mathbf{r} \):

\[
\frac{D\mathbf{r}}{Dt} = \mathbf{u}
\]

(4)

\( \mathbf{T} \) is the stress tensor of a Newtonian fluid:

\[
\mathbf{T} = (-p + \lambda \operatorname{tr} \mathbf{D}) \mathbf{I} + 2\mu \mathbf{D},
\]

(5)

with \( \mathbf{D} \) being the rate of strain tensor, i.e. \( \mathbf{D} = (\nabla \mathbf{u} + \nabla \mathbf{u}^T)/2 \); \( \mu \) and \( \lambda \) are the viscosity coefficients.

The equation of state 3 can take different forms in the SPH context [14].

### III. Numerical model

#### A. SPH model

In AQUAgpusph the following discretized version of the operators has been implemented [15], [16]

\[
\begin{align*}
\left\langle \frac{\nabla p}{\rho} \right\rangle_a &= \frac{1}{\gamma_a} \left( \sum_{b \in \text{fluid}} \frac{p_a}{\rho_a^2} + \frac{p_b}{\rho_b^2} \right) \nabla W_{ab} m_b \\
&\quad - \sum_{b \in \text{boundary}} \rho_b \left( \frac{p_a}{\rho_a^2} + \frac{p_b}{\rho_b^2} \right) W_{ab} n_b S_b \\
\left\langle \rho \operatorname{div}(\mathbf{u}) \right\rangle_a &= \frac{1}{\gamma_a} \left( \sum_{b \in \text{fluid}} (\mathbf{u}_b - \mathbf{u}_a) \cdot \nabla W_{ab} m_b \\
&\quad - \sum_{b \in \text{boundary}} \rho_b (\mathbf{u}_b - \mathbf{u}_a) \cdot n_b W_{ab} S_b \\
\left\langle \Delta \mathbf{u} \right\rangle_a &= \frac{\mu}{\rho_a \gamma_a} \left( \sum_{b \in \text{fluid}} K (\mathbf{u}_b - \mathbf{u}_a) \cdot \frac{\mathbf{x}_b - \mathbf{x}_a}{|\mathbf{x}_b - \mathbf{x}_a|^2} W_{ab} m_b \\
&\quad \sum_{b \in \text{boundary}} 2(\mathbf{u}_b - \mathbf{u}_a) \cdot \frac{(\mathbf{x}_b - \mathbf{x}_a) \cdot n_b}{|\mathbf{x}_b - \mathbf{x}_a|^2} W_{ab} S_b \right)
\end{align*}
\]

(6) 

(7) 

(8)

Where \( K \) is a parameter depending on the spatial dimension \( (K = 6, 8, 15, \text{respectively in } 1D, 2D \text{ and } 3D) \), and

\[
\gamma_a = \sum_{b \in \text{fluid}} W_{ab} m_b
\]

(9)

The Shephard correction factor of equation (9) has traditionally been removed from the SPH literature due to the fact that it breaks the symmetry of the operators, and should be approximately 1 for all particles far enough of the free surface (if fluid extensions are used in the solid boundaries). In AQUAgpusph, this term can optionally be computed.

Regarding the boundary terms in the operators (6), (7) and (8) have been introduced for the first time by De Lefèvre et al. [10], and formalized later by Ferrand et al. [11]. These terms can be used to impose the boundary conditions as will be described in the section III-B

#### B. Boundary conditions

1) **Free surface boundary conditions**: Free surface kinematic and dynamic boundary conditions are automatically accomplished in the WCSPH formulation. Considering null the ambient pressure \( p_0 \) in the EOS (3), consistency from an integral point of view can be demonstrated [16].

2) **Solid boundary conditions**: Since many other SPH alternatives are based on a specific solid boundary condition [5], [9], in AQUAgpusph most popular solid boundary conditions have been implemented, namely:

1. Fixed particles.
2. Virtual Ghost particles.
3. Elastic bounce.
4. Boundary integrals.

Solid boundary conditions will be discussed in section IV-C

#### C. Time numerical integration

In order to numerically approximate the solution of the problem, time integration is discretized as well.

One of the key features of the WCSPH model is that it is a purely explicit formulation. In AQUAgpusph a quasi-second order Leap-Frog method is used [17], only requiring one derivative computation per time step.

### IV. AQUAgpusph features

In this section, some details about the AQUAgpusph implementation will be discussed, focusing on the key points that differentiate it from other available SPH codes [5], [7]–[9].

From an algorithmic point of view, AQUAgpusph has been developed following the classic server-client structure represented in figure 1. In this structure several stages can be differentiated:

1) The initialization stage, where the input data is loaded and the simulation built.

2) The main loop, running while \( t < t_{\text{sim}} \), where \( t_{\text{sim}} \) is the maximum simulation time. At each iteration, the host calls the server in order to compute the solution until an output file is printed, or eventually until the end of the simulation is reached.

3) The server internal time loop, performing the time integration process during which no data must be sent to the host.

The first stage is associated with the “Setup” box of figure 1.

Regarding the third stage, which is the entirely involved with the server side, is the most computationally expensive and is the part of the code accelerated with OpenCL.
4.3 TFlops, compared to an NVidia GTX 680 card (∼300€), which is less powerful and 50% less expensive (GHz edition) card (∼450€) than the NVidia ones. For instance an AMD ATI HD 7970 approximately 50% more powerful and 50% less expensive than the NVidia ones. For instance an AMD ATI HD 7970 approximately 50% more powerful and 50% less expensive than the NVidia ones.

A. Free software

AQUAgpusph is free software, licensed under the GPLv3 terms. The users are therefore welcome to read, edit and redistribute the code, with the unique limitation that the redistributed versions must be shared with the same license.

B. OpenCL acceleration

As mentioned in the previous section, all the differential operators computation and the time integration is performed on the server side, which is the part of the code that is executed using OpenCL to accelerate the process.

The key features that lead to use the OpenCL instead of other alternatives like CUDA [1], OpenMP [2] or MPI [18] (all of them successfully used in other free available SPH implementations [5], [7]–[9]), can be summarized in the following list.

1) More powerful devices at lower costs: Using the CUDA framework, only NVidia devices can be used, while with OpenCL, other vendors devices can be applied, that may result in a really significant difference.

Actually AMD commercializes graphic devices that are approximately 50% more powerful and 50% less expensive than the NVidia ones. For instance an AMD ATI HD 7970 (GHz edition) card (∼300€), has a computational power of 4.3 TFlops, compared to an NVidia GTX 680 card (∼450€) with a computational capability of 3.1 TFlops.

These costs and processing differences are really volatile, being dependant on the general TIC market, but illustrate the importance of not restricting your developments to a unique vendor.

2) Hardware diversification: OpenCL is not restricted to GPUs, CPUs, or IBM Cell architecture, but is able to deal with all of them. This feature must be taken into consideration when a framework is selected, not only because the architecture may not be the best one, but also because OpenCL is easier to be adapt to new architectures that will be created in the future.

In addition to this, several developments are extending the OpenCL interface to work with clusters in a fully transparent way [19], [20].

Other SPH developments require 3 different versions of the software [4], [21], [22]

1) A serial CPU version.
2) A parallel CPU version.
3) A CUDA based version for GPUs.

With the OpenCL standard, a unique version of the code can cover all these versions, and moreover, it can do it simultaneously, allowing the use of GPUs and CPUs in the same simulation.

3) Highly modable application: OpenCL is based more on the GLSL working method, where the codes that will be executed in the computation device can be loaded and compiled in runtime, allowing to relay them to external files from the source code.

Of course you can create your OpenCL programs inside the main program source codes, but in the development of AQUAgpusph we found preferable to let the OpenCL source codes in external files, and allow the user to select which ones will be used in each simulation trough the input XML files.

C. Several boundary conditions implemented

As mentioned in the section III-B several solid boundary conditions have been implemented in AQUAgpusph in order to let the users choose the most appropriate for each situation.

1) Fixed particles: Fixed particles is a fluid extension based method where the fluid domain is extended placing a set of fluid particles on the other side of the solid boundary to which they are linked.

In figure 2, a schematic view of the fixed particles distribution is shown. The position \( r \), velocity \( u \), density \( \rho \) and pressure \( p \) fields are computed such that

\[
\langle r \rangle_a = r(R) \quad (10)
\]

\[
\frac{\partial \langle r \rangle_a}{\partial t} = -\rho_a \langle \text{div}(u) \rangle_a \quad (11)
\]

\[
\langle u \rangle_a = u(V) \quad (12)
\]

\[
\langle p \rangle_a = p(\rho_a) \quad (13)
\]

Where \( R \) and \( V \) are the wall position and velocity respectively. The fixed particles position and velocity is obtained directly from the solid motion, the density comes from an evolution process where the continuity equation (1) is used, and the pressure field is computed using the EOS (3).

2) Virtual Ghost particles: Ghost particles method was born as an improvement in the Fixed particles one, where the fluid extension is computed as a mirroring process of the fluid domain instead of a fixed set of particles.

In figure 3, a schematic view of the Ghost particles distribution is shown. Each fluid particle \( a \) produces a mirrored
ghost particle \( \hat{a} \), such that the fields can be obtained entirely from the fluid particle, and the boundary data

\[
\langle r \rangle_{\hat{a}} = r(\hat{a}, R) \tag{14}
\]
\[
\langle u \rangle_{\hat{a}} = u(\hat{a}, V) \tag{15}
\]
\[
\langle p \rangle_{\hat{a}} = p(\hat{a}, g, r_{\hat{a}}, R) \tag{16}
\]
\[
\langle \rho \rangle_{\hat{a}} = \rho(\langle p \rangle_{\hat{a}}) \tag{17}
\]

Several methods can be selected to perform the extensions of the velocity and pressure [23].

Since the number of Ghost particles is not known in each time step, this method doesn’t quite fit in GPU based codes. For this reason, in AQUAgpusph, a slightly modified version called Virtual Ghost Particles has been implemented.

The Virtual Ghost Particles avoid the need to store the Ghost particles positions and fields before computing the differential operators approximation. In this approach, the differential operators computation process is performed once for each solid wall and for each particle near enough to the solid boundary, all neighbouring particles are mirrored in order to compute them.

This new approach has the disadvantage of being less efficient than the usual way due to the fact that for each particle and boundary, the mirroring process must be done more than one time. On the other hand, this method better fits the Takeda extension model [24].

3) Elastic bounce: The simplest solid boundary condition is the elastic bounce one. The elastic bounce only computes the force necessary to prevent particles trespassing a solid wall, such that at the final of the time step

\[
\langle u \rangle_{\hat{a}} \cdot n = (V - \mu_e u_{\hat{a}}) \cdot n \tag{18}
\]

Where \( n \) is the normal of the solid wall, and \( \mu_e \) the elastic bounce factor, \( \mu_e = 1 \) corresponds to a fully conservative interaction), as shown in figure 4.

This boundary condition has not been designed to work alone but along with other boundary conditions in order to avoid a solid penetrating.

4) Boundary integrals: The Boundary integrals method consist in retaining the boundary terms of the equations (6), (7) and (8) [10], [11].

In figure 5, a schematic view of this boundary condition method is shown. The fields in the area elements is computed in a similar way than the Fixed particles model.

\[
\langle r \rangle_a = r(R) \tag{19}
\]
\[
\langle u \rangle_a = u(V) \tag{20}
\]
\[
\langle p \rangle_a = \frac{1}{\gamma_a} \left( \sum_{b \in \text{Fluid}} \left( \frac{p_b}{\rho_b} - g \cdot r_{ab} \right) W_{ab} m_b \right) \tag{21}
\]
\[
\langle \rho \rangle_a = \rho(\langle p \rangle_a) \tag{22}
\]

While the position and the velocity of the boundary elements are linked to the solid motion, pressure and density are obtained from the fluid data. The main difference in the fields computation along the boundary with respect to the Fixed particles technique, is that the pressure and density fields doesn’t come from a time integration but are interpolated from fluid data for each time step.

D. Input data in XML files

AQUAgpusph uses an XML file based interface, allowing for the creation of structured input data distributed in several files.

Also, being the XML files plain text, they can be generated from external applications, mandatory to provide a scriptable application.

E. Python extensible

Python scripting has been integrated in the solids motion interface.

Python is a script based language that has become really popular in the last years due to its simplicity, power and increasing support. It is a free language that does not require
a private platform to be executed, and has been implemented in the most used operating systems.

Python scripted motions are oriented to be able to execute more complex motions, such as coupled ones, where the motion data is not previously known, i.e. case number 9 of the SPHERIC benchmark, which has been simulated with AQUAgpusph [13].

This feature, combined with the highly modable one derived from the OpenCL usage (see IV-B3), allows for the user to create customizable motions without a significant performance lost. Figure 6 depict the working methodology of the Python controlled motions. 3 main stages can be described:

1) The setup, that recollects some simulation data such as the simulation time, the time step, and the fluid force and moment, in order to pass it to the Python script.
2) The Python script execution. Python execution must return the motion definition data (Quaternion data, transformation matrix, ...).
3) The OpenCL script execution. OpenCL will modify the particles according to the received data from the Python script.

Therefore, with this methodology, the general behaviour can be controlled with a Python script, and more control over each particle can be achieved through a custom OpenCL script.

F. Some validations have been carried out

Several scientific works have used AQUAgpusph to perform simulations, validating the tool therefore.

In the section V, a test case with the computed results using AQUAgpusph will be discussed.

V. Results

A. Test case definition

In this paper, case C described in [25], has been simulated. This case has been selected in order to have experimental data and results from a previous SPH solver as well.

A rectangular tank of 90 x 58 x 5 cm, and a water depth of 9.3 cm, is excited via an electrical motor with a sinusoidal type rotating motion whose amplitude is $\theta_{max} = 4^o$ and with different excitation frequencies. In case C, an excitation period of 0.9 times the first sloshing one is selected, where the highest pressure peaks were registered.

In figure 7, a schematic view of the experimental setup is shown. In this case, we focus on the pressure registered from sensor 1.

B. Results comparison

This case has been simulated using both a GPU AMD HD-7970 (2048 processors) and a CPU AMD FX-8120 (8 cores). The performance associated with each device will be discussed in the section V-C.

In figure 8, the pressure field for the first wave impact is shown. The acoustic waves are clearly visible but no instabilities arise from the boundaries, with an acceptable level of noise.

The pressure interpolated at sensor 1 has been plotted in figure 9, along with the pressure registered experimentally and with the results of the simulation with a previous SPH solver.

Both simulations have been performed with the same Reynolds number, but in this case to impose the solid boundary conditions the boundary integrals [10], [11] has been applied, instead of the Ghost particles used before [25].

New simulations better reproduce the size of the pressure peak with a pressure signal noise significantly reduced.
C. Performance

As mentioned above, two different devices have been used to perform this simulation. In table I the devices features are described. Note that the GPU is approximately 50 times more powerful than the CPU.

In SPH, the time consumed by simulations mainly depends on two parameters, the number of particles \( n \), and the number of neighbours per particle, which can be described by the parameter \( dx/h \), where \( dx \) is the distance between particles.

Figure 10 shows the time consumed to compute 1 time step depending on the number of particles \( n \), for a fixed number of neighbours. Note that the time consumed with the GPU has been multiplied by a factor of 30. Therefore, the GPU has computed each time step \( \sim 30 \) times faster, in front of the powerful rate of 50 depicted by the table I (60% of efficiency).

The values of the time consumed per time step in computations are of the same order than the described in [4]. Further analysis can’t be carried out due to the \( dx/h \) ratio not being documented (from the case setup and the documented number of particles and \( h \) length, \( dx/h = 0.67 \) ratio can be expected).

Regarding the scalability, let us define the parameter

\[
S_n = \frac{\partial \Delta T}{\partial n}
\]  

(23)

Such that if \( S_n = 1 \) the best desirable scalability has been achieved, so the time consumed per time step computation increases on the same rate than the number of particles. A scalability parameter greater than 1 causes some performance to be lost when the number of particles is increased, and scalability parameters lowers than 1 means that some performance can be gained through increasing the number of particles.

In this case, we have the following scalability parameters

\[
S_{n_{CPU}} = 1.07 \quad S_{n_{GPU}} = 0.78
\]  

(24)

The CPU execution shows a great optimization level, but in the case of the GPU, it can be noticed that in the studied range of \( n \), the GPU can heavily benefit from a number of particles increase.

In figure 11 the time consumed to perform one simulation time step is also represented, but in this case it is plotted as a function of the parameter \( \frac{1}{n \cdot dx} \), leaving the number of particles \( n = 200000 \).

The GPU has \( \sim 30 \) times more processing power than the CPU again. Let us define the scalability for this case as

\[
S_{dx/h} = \frac{\partial \Delta T}{\partial (h/dx)}
\]  

(25)

We find that the scalability is dramatically different up to \( dx/h = 0.25 \)

\[
S_{CPU_{dx/h}} = \begin{cases} 
0.97; & \text{if } dx/h > 0.25 \\
3.2; & \text{otherwise} 
\end{cases} 
\]

\[
S_{GPU_{dx/h}} = \begin{cases} 
0.44; & \text{if } dx/h > 0.25 \\
1.72; & \text{otherwise} 
\end{cases} 
\]  

(26)

So in this case, it seems that a best performance point can be found for \( dx/h \sim 0.25 \) for both the GPU and CPU, and therefore the code is optimized for the range of number of neighbours considered.

VI. Conclusions

In this paper AQUAgpusph, a new free SPH tool, has been described. This tool is in the state of the art of the SPH GPU based codes, and includes features that have been discussed in section IV, which are not present in other available SPH implementations.
AQUAgpusph has been designed in order to provide a valuable scientific tool useful for 3rd parties and designed as a modable and extensible application.

Additionally, some results showing the great capabilities of the new code compared with the previous one has been discussed, showing that the new features can be beneficially applied. The performance has also been discussed.

AQUAgpusph future work developments may comprise at least the following ones:

1) Simultaneous multi-device computation support. Using some of the new developments like [19], [20], the code can easily be parallelized in order to simultaneously use several devices in the way described in [5].

2) AQUAgpusph is highly modable but some improvements can be done, e.g. allowing for more flexible input parameters.

3) Python interface must be extended to other parts of the code, integrating pyopencl to better simulations customization.

4) Double precision must be supported. For now, the double precision capabilities of the GPU devices is significantly lower than the single precision ones, but this tendency has been changing in recent years.

5) Truly compressible SPH model can be implemented in order to compare it with the weakly compressible one.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the Spanish Ministry for Science and Innovation under grant TRA2010-16988 “Caracterización Numérica y Experimental de las Cargas Fluido-Dinámicas en el transporte de Gas Licuado”.

The authors are grateful to Hugo Gee for the correction and improvement of the English text.

REFERENCES


[11] M. Ferrand, D. R. Laurence, B. D. Rogers, D. Violeau, and C. Kassiotis, “AQUAgpusph future work developments may comprise at least the following ones:

1) Simultaneous multi-device computation support. Using some of the new developments like [19], [20], the code can easily be parallelized in order to simultaneously use several devices in the way described in [5].

2) AQUAgpusph is highly modable but some improvements can be done, e.g. allowing for more flexible input parameters.

3) Python interface must be extended to other parts of the code, integrating pyopencl to better simulations customization.

4) Double precision must be supported. For now, the double precision capabilities of the GPU devices is significantly lower than the single precision ones, but this tendency has been changing in recent years.

5) Truly compressible SPH model can be implemented in order to compare it with the weakly compressible one.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the Spanish Ministry for Science and Innovation under grant TRA2010-16988 “Caracterización Numérica y Experimental de las Cargas Fluido-Dinámicas en el transporte de Gas Licuado”.

The authors are grateful to Hugo Gee for the correction and improvement of the English text.

REFERENCES


[11] M. Ferrand, D. R. Laurence, B. D. Rogers, D. Violeau, and C. Kassiotis, “AQUAgpusph future work developments may comprise at least the following ones:

1) Simultaneous multi-device computation support. Using some of the new developments like [19], [20], the code can easily be parallelized in order to simultaneously use several devices in the way described in [5].

2) AQUAgpusph is highly modable but some improvements can be done, e.g. allowing for more flexible input parameters.

3) Python interface must be extended to other parts of the code, integrating pyopencl to better simulations customization.

4) Double precision must be supported. For now, the double precision capabilities of the GPU devices is significantly lower than the single precision ones, but this tendency has been changing in recent years.

5) Truly compressible SPH model can be implemented in order to compare it with the weakly compressible one.

