Real-time simulation of highly-viscous fluids using the Fluid Redistribution Forecasting (FRF) algorithm

by Federico Coletto

Modern-day technologies enable videogame programmers to implement a number of graphical effects, amazing IAs and features that until just a few years ago would have been considered pure fantasy.
One of the crucial aspects of designing video programs for games consists in achieving the highest possible interaction between the player and the simulated environment that must be realistic enough to be both engrossing and engaging.
To be realistic, the environment must first of all enable the user to credibly influence the objects it includes, that in turn must be made to correctly interact with one another.
The brand new ***FRF (Fluid Redistribution Forecasting)*** algorithm I developed, will try to satisfy this kind of requests constituting a solid development base.

Table of contents:

Scope

The scope of this work is to demonstrate the efficacy of the ***FRF algorithm***, creating a system that will enable dynamic and credible interaction of fluid masses with the simulated three-dimensional environment.
Imagine a potential game situation where the character has to flee from a flow of lava streaming down from a volcano, or a candle that melts under the heat of its own flame.
Or imagine, for example, that you are caught up and drowning in an enormous tank full of glue, or that you have shrunk and can see giant drops of resin running down tree trunks.
All these conditions require the right interaction with these masses of fluid and, above all, require these fluids to behave consistently in consideration of the force of gravity, the shape of the surface they are sliding along, their density, mass and viscosity.
The first requirement, the player's interaction with the masses, will not be discussed in this paper as it is strictly connected to the specific context it occurs in, namely the type of game and its basic concept.
What we will focus on is a more intriguing aspect, at least from the point of view of physics and simulation: the interaction of the masses with the game environment, designing a suitable architecture for a strongly object-oriented development, simulation and rendering, to make it as flexible, recyclable and customisable as possible.

1. Introductory notes

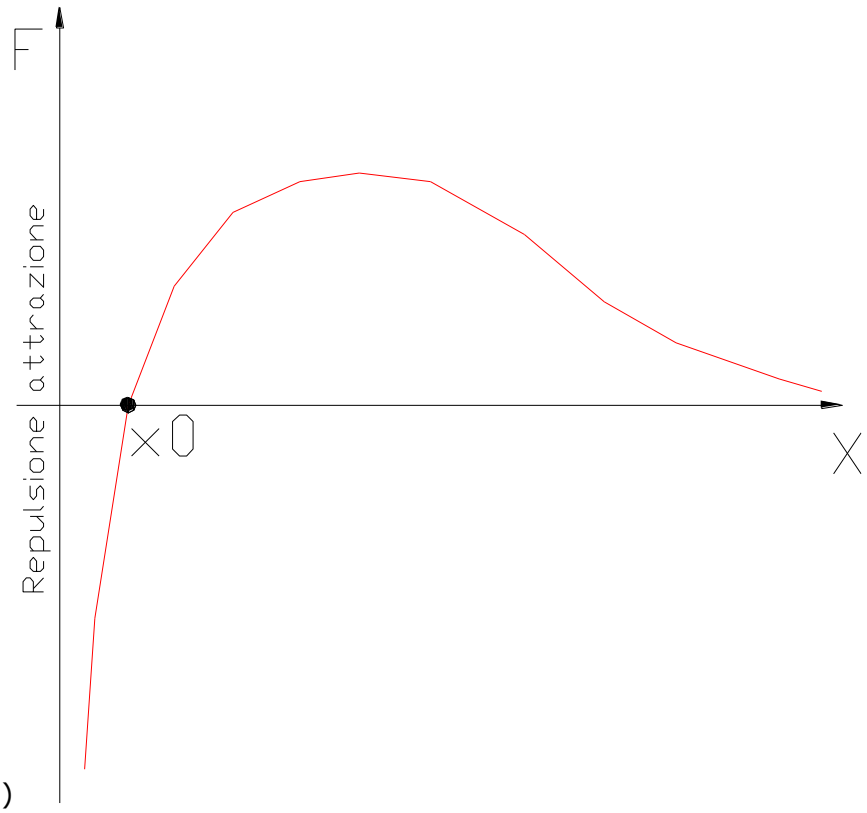
As stated above, we need a software based on the ***FRF algorithm*** that can be programmed to be recycled and integrated in the future: consequently reference will be made to flexible design patterns that are suited to the final aim.
In view of the "experimental" nature of the project, we shall not linger on low-level code optimisation processes, and will try to adopt a more easily readable and human-friendly approach as possible.
The optimisation strategies can very well be designed during the refactoring phase (see XP programming) if it is decided to integrate it in different host systems or when considering the its possible release on a specific hardware target/range.
Starting from zero, a framework will have to be built that will act as the project's backbone, in addition to services for managing models, textures, user input. Structures and methods will also have to be envisaged for the necessary physical and mathematical computations, whereas others will be dedicated to geometrical rendering.
The design of the whole engine will be viewed through UML diagrams.
The basic knowledge needed to understand this document and related sources include:

- the fundamentals of software and design patterns programming
- C++ and OOP (object-oriented programming)
- UML (unified modeling language)
- OpenGL, vector analysis, dynamics, force and motion.

In order to face the different aspects of the topic without any hesitation whatsoever, we will proceed to providing a wide initial outlook on the fundamental principles that govern dynamics and the different aspects of fluid mechanics.
Only when it is time for implementation will we choose the most suitable amongst the exposed relations, and dedicated ones will be formulated in order to solve specific issues.
This will enable a full understanding of what is really useful in view of our objective and what can be left out or approximated whenever expressly required by the theory of physics.
Each critical or important item of the project shall in any case be exposed trying to enable its full comprehension even by those who are not especially familiar with the topics in question, and shall be allocated a progressive "(n)" number.
The units of measure used in the equations are part of the MKS system (metre-kilogram-second).

2. Intermolecular forces and the states of matter

First of all, a pre-emptive survey will be needed to help understand exactly what fluids are, how they behave and why.
This will make it easier to understand the concepts that will be explained anon, and will enable us to provide a definition of a simulation model.
Like all material objects, fluids are made of material particles (atoms or molecules).
In all bodies, the particles are interrelated by forces of attraction/repulsion that are called "intermolecular forces", the nature of which is basically electric (as in this case the gravitational components are almost entirely negligible).
Intermolecular forces act to keep the body's molecules at a so-called bond distance, that equals the diameter of the molecule.
In the case of two interacting molecules, the forces are attractive when their reciprocal distance is higher than the bond distance (called cohesive forces in same-kind molecules, adhesive forces if they are different), and repulsive when it is lower.
This case is represented in graph (1). *F* is the exerted force, *X* is the distance between the molecules' barycentres, *x0* is the bond distance.



(fig.1)

It is quite simple therefore to grasp why matter can appear in different 4 states:

1. ***Solid matter***: has a definite shape and volume. Intermolecular cohesive forces are very strong and the particles are spatially arranged in a crystalline structure. Solids are not at all compressible or elastic.
2. ***Liquid matter***: while it has a definite volume, it does not have a definite shape and in the absence of gravity, external forces and/or stress it tends to take a spherical shape (this will be analysed in greater depth anon). This occurs because the cohesive forces are much weaker than in solids, and the molecules are free to "slide" on each other while maintaining the same total volume. Like solids, liquids are not particularly compressible.
3. ***Aeriform matter***: it has neither a definite shape nor volume, as its cohesive forces are practically inexistent providing particles with an enormous freedom of movement. Gases are extremely compressible.
4. ***Plasma***: atom structures are dispersed and the existing temperatures are very high.

3. Surface tension

Before we continue, it is essential to understand the behaviour of a liquid in an environment where the force of gravity is not equal to zero: this will also shed some light on the reason why liquids tend to take on a spherical shape when there are no external forces.
Consider (with gravity) a liquid in a container and the surface layer of the molecules the thickness of which is less than the radius of the sphere of action of the intermolecular forces: it is simple to perceive that the same molecules tend to establish a balanced environment perpendicular to the force of gravity (abbr. f.o.g.).
This however is not true longitudinally to the f.o.g., signifying that each molecule tends to be dragged downwards, that is in the direction of the same f.o.g..
However, from what we have learned on the liquid state of matter, it is clear that the volume has to remain constant and as the current environment is a container the same goes for the number of molecules per surface unit.
This leads to the conclusion that the liquid will tend to minimise the surface area, taking on a shape that, with an equal volume, will have the smallest external surface possible.
It is self-evident that without any f.o.g. this shape will be that of a sphere, whereas when the f.o.g. is present it will be flat.
Concluding, the surface tension can be defined as being the contraction force that acts on each metre of the contour line of a fluid's free surface.
It can be represented by the relation:

(0) $\tau = \frac{F}{l}$

where τ is the surface tension constant, whose value depends on the liquid in question, F is the contraction force and l is the length of the surface line.

4. Internal friction and viscosity

An important feature of bodies (but especially of fluids) is their internal friction, that is the resistance that a fluid's molecules meet when sliding on each other.
This resistance is a force that describes the behaviour of the dragging of a fluid's layers, which is defined based on the distance between the layers that have been considered, the current surface and speed.
The stated relation is represented by the following equation:

(1) $F = \eta \frac{S\Delta v}{d}$

where F is precisely the dragging resistance between two layers of a same fluid that are distant d and with a slipping speed Δv.
In the formula, η represents the *viscosity coefficient*, the value of which depends on the fluid in use.
The dimensional formula of **(1)** is

(2) $[\eta] = [l^{-1} m t^{-1}]$

which leads to a simple definition of viscosity, that can be imagined as being the dragging force between two fluid layers of an area of $1m^2$, at a distance of 1m, the speeds of which differ by 1m/s.
These specifications lead to the conclusion that a highly-viscous fluid will have relatively strong molecular bindings: consequently it will tend to oppose greater resistance to changing shape due to external forces (such as, for example, force of gravity).
This proves that extremely viscose fluids will find it hard to slide on surfaces, will tend to adhere and are unlikely to cause vorticity effects.

5. Preliminary notes on dynamics

Before delving into the main issues of the topic, it is worth mentioning some basic notions of physics and dynamics that may be a useful reference to understand the illustrated concepts and to design the simulation models.
In physics there are three fundamental principles that describe the behaviour (motion) of bodies based on the forces exerted on it.
These principles were first discovered and described by Isaac Newton and are known as the "*Three fundamental laws of dynamics*".

(3) The first law of dynamics:
"*Every body remains at rest or in a state of uniform rectilinear motion unless it is subject to a force other than zero*"

This simply means that a body does not move, accelerate, decelerate or change unless suitable forces are applied to it.

(4) Second law of dynamics (or fundamental law of dynamics):
"By applying a force $\vec{F}$ *to a free body, the acceleration* $\vec{a}$ *of the latter has the same the direction and angle as the applied force and its magnitude is directly proportional to the force magnitude."*
This leads us to the relation

(5) $$\frac{\vec{F}}{\vec{a}} = k$$

Where "*k*" is a constant representing the body in question's *mass of inertia*, namely the resistance it opposes to the actions that tend to vary its velocity.
Conventionally "k" is replaced by "m", a symbol that universally indicates a mass.
Relation **(5)** hence becomes

(6) $$\frac{\vec{F}}{\vec{a}} = m$$

which can also be expressed as

(7) $\vec{F} = m * \vec{a}$

having the dimensional formula

$[F] = [mlt^{-2}]$ indeed 1N = 1Kg * 1m/s^2

It is now worth making a digression that will surely prove useful later on.
Considering a body at rest having mass "m", based on the second principle of dynamics the force it exerts downwards equals

$$\vec{F} = m * \vec{a}$$

where, however, $\vec{a}$ represents the gravitational acceleration that is commonly expressed through the constant $\vec{g}$ (equalling 9.81 m/s^2 at sea level and at a latitude of 0°) and $\vec{F}$ is the force-weight in question, represented by the symbol $\vec{P}$.
Hence, **(7)** becomes

(8) $\vec{P} = m * \vec{g}$

(9) The third law of dynamics:
"*Every action is contrasted by an equal and opposite reaction*".

This means that reciprocal actions, namely the force between two interacting bodies, are always equal in magnitude and opposite in direction.
Imagine for example two bodies called "A" and "B".
(9) indicates that if (for example) A acts on B, then A experiences an equal and opposite reaction from the same B.
This event is described by the relation

(10) $\vec{F}_{ab} = -\vec{F}_{ba}$

6. The fundamentals of fluid mechanics

We previously introduced a number of basic concepts relating to the behaviour of the molecules of bodies according to their cohesive forces.
We will now take a look at some of the macroscopic effects that these forces have in environments in which the force of gravity is significant.
Not all these examples and principles will be directly useful for the final modeling, but all will be very helpful to understand and forecast the behaviour of the fluids from a general and also from a more specific viewpoint.
As the masses the behaviour of which we want simulate are fluids, we need consider the basic concepts of the specific branch of physics that studies their behaviour: *fluid mechanics*.
Fluids mechanics focuses on liquid and gaseous substances from both a static and a dynamic viewpoint and has been applied to numerous and varying fields including meteorology, aeronautics, naval engineering, mechanics and chemistry.
Within this field we can basically identify two branches.
The first, *fluid statics*, studies the equilibrium conditions of fluids at rest.
It can be divided into the *statics of gases* and *hydrostatics* according to the features of the analysed fluid.
Fluid dynamics, the second branch, can, just like the former, be divided in another two branches: *aerodynamics* (or dynamics of gases) and *hydrodynamics*, respectively focussing on moving gases and liquids.
At this stage, it is essential to introduce a number of parameters that must necessarily be considered when facing their rigorous application for the sake of a simulation and that should not be disregarded even when the application has a more qualitative rather than quantitative nature.
The concept of *pressure* (symbol "*p*"), often arises in fluid mechanics, and is closely connected to the concept of *force*.
It does in fact represent force "F" acting on the surface unit "*S*" normal to the direction of the same force and represented by the relation

(11) $p = \frac{F}{S}$ the dimensional formula of which is $[l^{-1}mt^{-2}]$

It is essential to emphasise that the pressure can be represented by several different units of measure, although the most commonly employed is

(12) the *technical atmosphere* (at) = $\frac{Kg}{cm^2}$

Its correspondent in the CGS system is

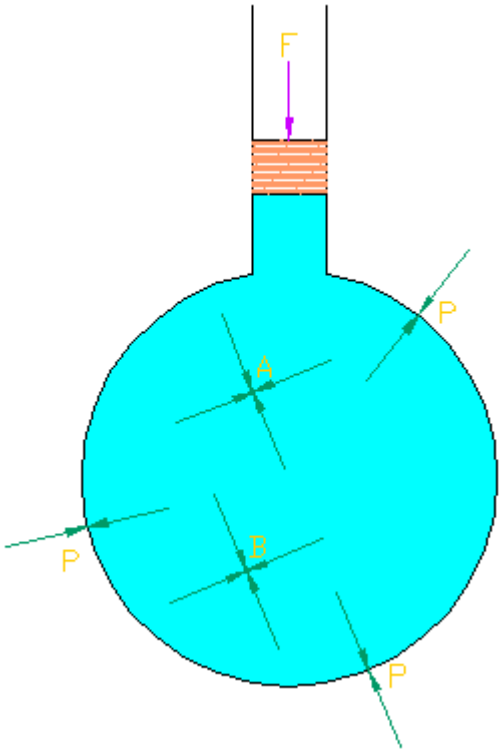
(13) $\text{baria} = \frac{\text{dine}}{\text{cm}^2}$

which is however often replaced by bar multiples (1 bar = 10^6 barye) as it is extremely small in size and therefore awkward to use.

7. The basics of fluid statics

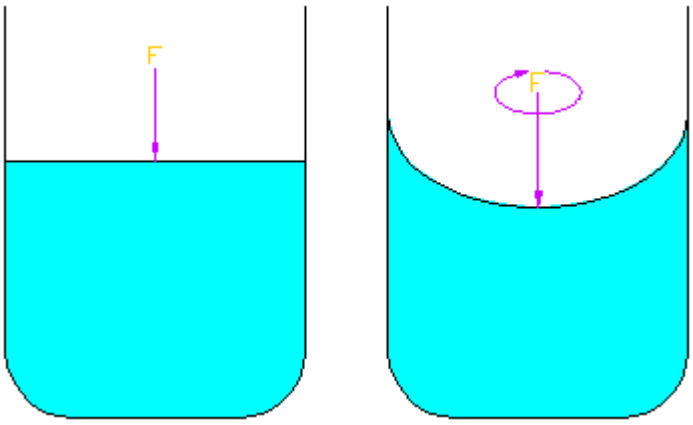
As was mentioned above, fluid statics focus on fluids at rest.

(14) One of the main features of a fluid in this state is that the force exerted on the particles it is made of (called *internal force*) has the same magnitude in all directions.
This can easily be explained and the explanation can easily be inferred by deduction.
If we were to take, for example, a case in which the aforementioned forces were different, then each particle would move in the direction of the resultant of the same forces.
This would result in the fluid not being at rest.
The concept can easily be extended to the fact that a liquid held in a container of an arbitrary shape exerts a system of forces tangential to the sides of the container the resultant of which is null.
Hence, the force exerted by the fluid on the sides is perpendicular to the latter and have the same intensity per area unit.
This basic rule was announced for the first time in 1647 by *Blaise Pascal* (French mathematician and philosopher) and the only condition required to confirm its validity is to disregard the pressure exerted by the force-weight of the fluid mass (condition that had been foreseen in *Pascal*'s original enunciation).
Drawing (2) graphically shows its behaviour.



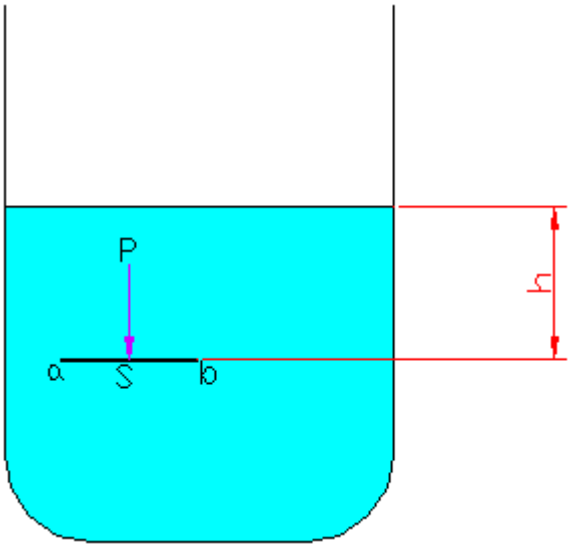
(fig.2)

(15) Another important parameter to consider in liquids at rest is the liquid's free surface that is always perpendicular to the resultant of the forces acting on it.
When a liquid is subject to gravity alone (in a vacuum and totally at rest) the free surface is flat and perfectly horizontal, whereas it tends to take a paraboloidal shape if the liquid is made to rotate in the vessel due a centrifugal force.
This behaviour is highlighted by drawing (3).



(fig.3)

(16) Based on the two illustrated principles, it is possible to infer the so-called "*Stevino Law*" whereby if we consider a liquid that is solely subject to the force of gravity, the hydrostatic (or aerostatic) pressure at any point inside the analysed liquid is directly proportional to the force/weight of the liquid-column overhead.
This simple behaviour can be better understood by referring to drawing (4).



(fig.4)

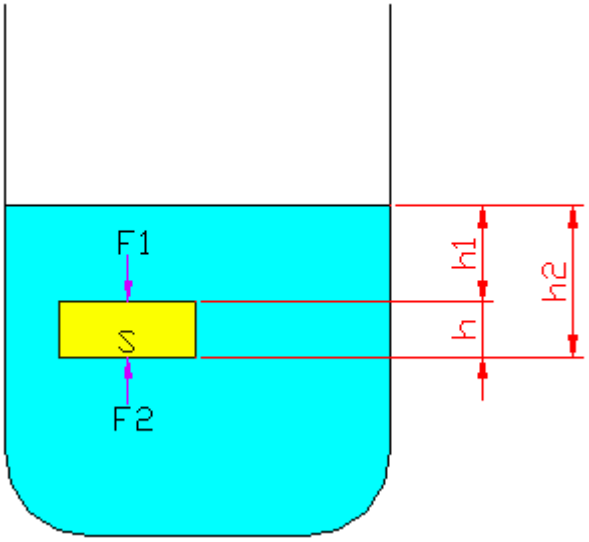
The Stevino Law is represented by

(17) $P = \gamma * S * h$ where γ is the fluid's absolute specific weight

Hence we can obtain the pressure "*p*" exerted on the surface "*S*"

(18) $p = \frac{\gamma * S * h}{S} = \gamma * h$

(19) Another rule that requires some consideration is the so-called "*Archimedean principle*", also known as the "*second principle of fluid statics*" whereby a body immersed in a fluid is subject to a down-upwards thrust (called "*buoyancy*") the magnitude of which equals the force/weight exerted by the volume of fluid it displaces.
As can easily be perceived, this law enables us to forecast the behaviour of floating bodies (or, of course, sinking bodies).
This concept is described in figure (5).



(fig.5)

This is easy to prove.
Take a body with a surface "S" immersed in a liquid.
The hydrostatic pressure from the side are balanced, as can be inferred very simply through the Pascal Law.
On the contrary, the pressure on the bases exerted by F1 and F2 are not equal due to the difference in volume and specific weight between the body and the fluid.
The hydrostatic pressure exerted downwards is

$p_1 = \gamma * h_1$ whereas the upwards exerted pressure is $p_2 = \gamma * h_2$

These two relations result in

$$\vec{F}_1 = p_1 * S = \gamma * h_1 * S$$
$$\vec{F}_2 = p_2 * S = \gamma * h_2 * S$$

The resultant "F" of force F_1 and F_2 will be

$$\vec{F} = \vec{F}_2 - \vec{F}_1 = \gamma * S * h$$

however, due to the volume "V" of the body being equal to

$V = S * h$ then $F = \gamma * V$

If we know that the weight of the body is

$P = \gamma * V$ then $F = P$

and considering that F and P have opposite directions

$$\vec{F} = -\vec{P}$$

8. The basics of fluid dynamics

Fluid dynamics is an extremely complex and mathematically interesting field of study: it enables to forecast the behaviour of fluids in the most disparate conditions of motion and combines concepts of dynamics, traditional mechanics and chemistry.
The problems relating to fluid dynamics issues that need to be resolved with a relatively small margin of error have to be faced by means of extremely "*time & memory critical*" methods and algorithms: this means that they are very time consuming from the viewpoint of the hardware resources employed for their processing, in terms of moth memory and machine cycles.
This is justified by the fact that the relations that describe the behaviour of fluids in motion are derivative/integral in nature and also because the particle-type models that are often resorted to require an amount of relational computations that can be in the order of n^2 (where n = number of particles in the model).
In history, the first significant steps forward in fluid dynamics were made by *Evangelista Torricelli* (the father of the barometer) who in 1643 formulated an important law that still bears his name.
The Torricelli law describes the relation between the velocity of efflux of a liquid from an orifice in the container it is contained in and the height of the level of the liquid itself above the aforementioned orifice.
Further steps forward were later made by *Leonhard Euler* who managed to apply the three laws of dynamics previously enunciated by *Isaac Newton* to fluids.
It is thanks to the above that it will be unnecessary to linger long on the basic concepts of fluid dynamics because (as we will see anon) the behaviour of the fluids we intend to simulate does not justify the use of the specific relations pertaining to this branch in order to achieve a rather *credible* simulation.
What will emerge is that the notions of dynamics and fluid statics will be more than sufficient to model the behaviour of a highly-viscous mass using the ***FRF algorithm.***

9. The idea and the basic model

Following this brief introduction on the properties and the behaviour of fluids, we shall focus on the core of our study: the determination of the features that the simulation model must hold.
In order to do so, we shall refer to the principles mentioned above or suitable approximations that will enable us to increase the efficiency of the program's implementation.
The first step consists in establishing the rules and restrictions that a hypothetical model must comply to.

- The motion of the mass must be determined by the combined forces acting upon it, including cohesive and adhesion forces;
- The shape of the fluid mass in absence of external forces shall be spherical;
- The free-falling mass in the direction oft he f.o.g. shall take on a drip shape;
- The shape shall be consistent with the combined forces acting on it and the shape of the surface it lies on, be it a container, a flat top or a arbitrarily curved surface;
- At all times, the volume must be constant;
- The surface shall be reduced to a minimum;
- The mass shall suitably respond to collisions, correctly managing the magnitude of motion, impulse, speed, acceleration, etc.
- The vorticity and all other reasonably arising phenomena linked to fluid dynamics must also be suitably managed.

Simultaneous compliance to all these restrictions is extremely complex and requires an amount of prohibitive hardware resources if the aim is real-time application.
By resizing our expectations, we can establish a set of restrictions that can be complied to in real-time by what is today considered as being average-high range PC.

a) The mass "at rest" (in absence of gravity and support surfaces) will be shaped like a "meatball", to enable its geometry to be easily changed.
b) The mass must suitably respond to the gravitational force, sliding down sloping surfaces and changing its shape to suit.
c) The shape of the mass shall always be rather rounded, to suit the surface tension and the viscosity that hinder its quick flow down the support surfaces.

d) Its volume must be kept as constant as possible.
e) The distribution of the volume of the mass must be determined based on the shape of the support surface.

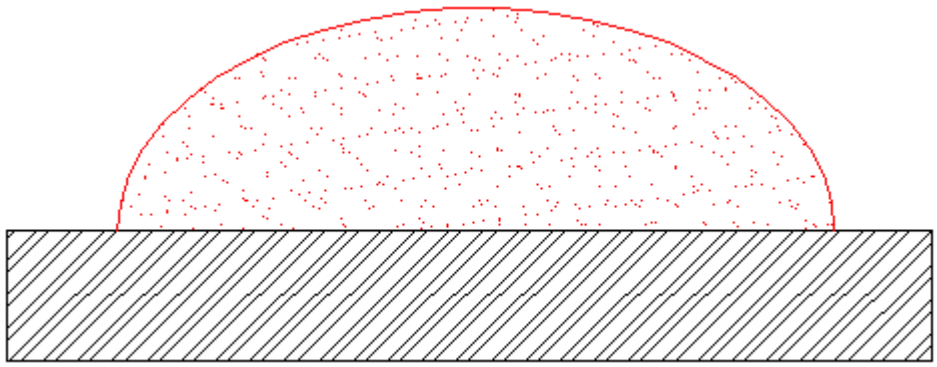
Hence, all behaviours and features that do not significantly contribute to the simulation of highly-viscose fluids in non-critical conditions will be ignored, i.e. with acting forces and surfaces that do not cause their fragmentation into several unconnected bodies.
Some of the main features that will be ignored for the sake of simplification are:

- Vorticity.
- Forces and surfaces that would fragment the initial mass into several unconnected bodies.
- Elastic reactions to collisions.
- Temperature changes.
- Rotating movements that could excessively and destructively distort the initial geometric structure.
- Surface minimization.
- Absolutely constant volume.

By analysing the restrictions, it is easy to establish that the mass could possibly be represented based on a polygonal model outlining its surface, where the position of the vertexes are recalculated for in frame so as to comply with the restrictions.

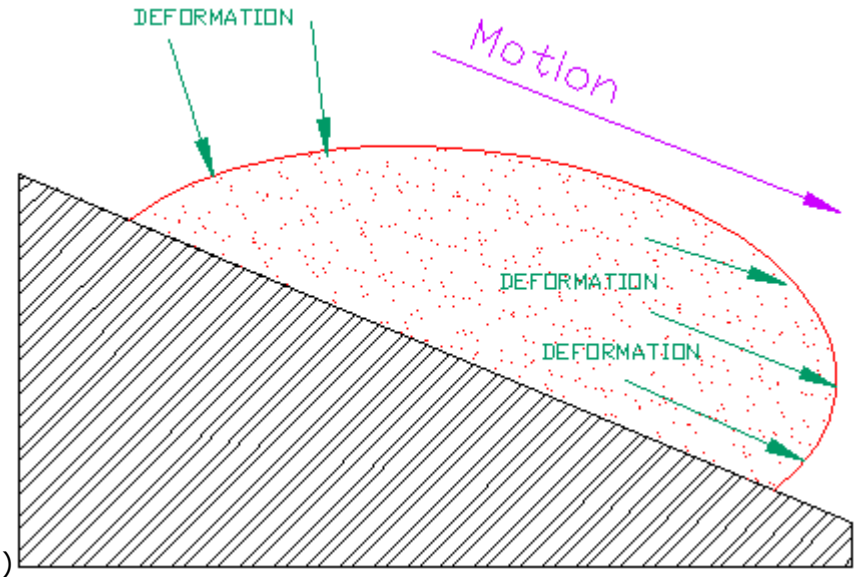
10. Case studies

Once the basic behaviour of the fluid body has been established, it is possible to create a set of representations of cases and typical interactions. This is useful to perceive the applicational "nuances" that will be encountered and that need to be analysed in view of their full and reliable implementation.
Let us start with an extremely simple case, that is a mass at rest on a horizontal surface.
Figure (6) shows the shape it will take based on the afore listed principles.



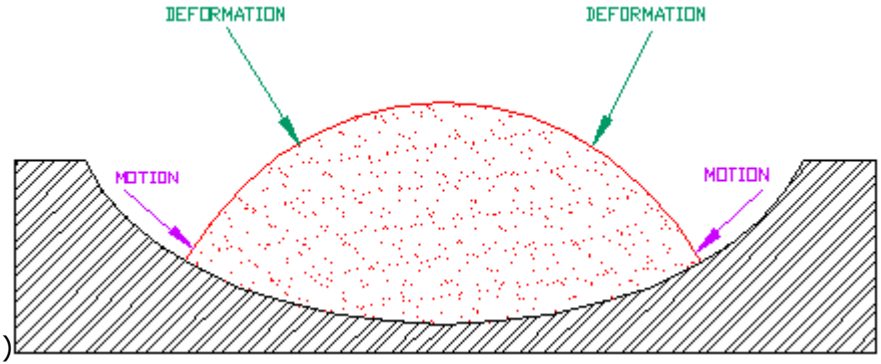
(fig.6)

A more complex case is represented by the same mass located on a slanting plane, the shape of which changes due to the redistribution of its volume and the uniformly accelerating motion imposed by the force of gravity.
This is how it would be represented (figure 7).



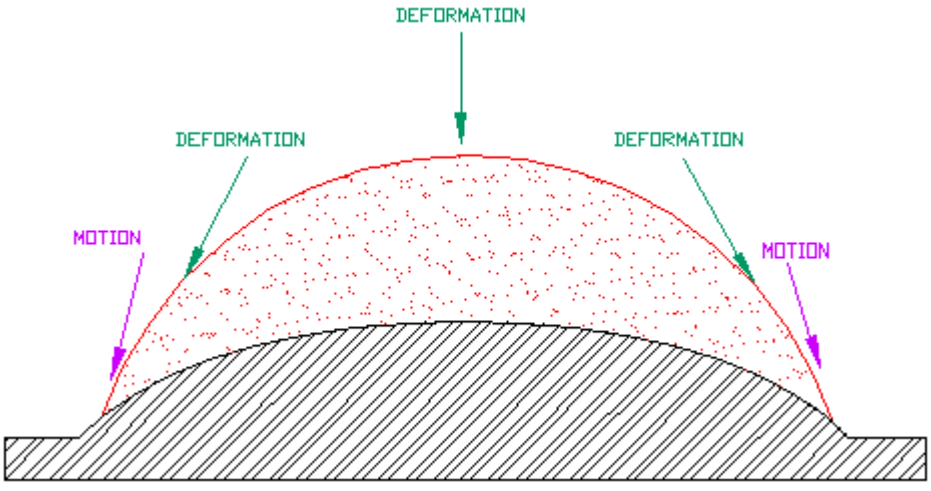
(fig.7)

If, instead, the mass were placed on a concave curved surface (see figure 8), we would expect its profile to change similarly to what we see in this figure.



(fig.8)

In a convex curve (figure 9) however it would change as follows

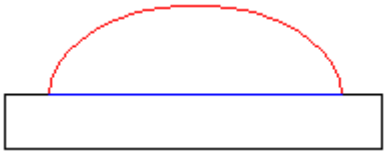


(fig.9)

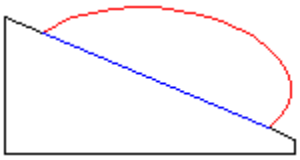
These examples easily prove that the body has a set of "implicit" features that were not listed above.
We will now see that these features can be exploited to our benefit to simplify the implementation of the simulation.
Because for now the simulation of falling bodies is not an issue, it is intuitively clear that part of the fluid's surface is always "hidden" and in contact with the slip plane, whereas part of it is "visible".
As the fluid moves and changes, these two parts also change, thereby redefining its shape.

All we have to do now is to simulate the evolution of the outer surface. As its geometrical shape is defined by the two aforementioned parts, it is possible to write down the relations that will enable quite a simplified calculation of its aspect.
These two parts of the fluid's surface shall be called SC (contact surface) and SV (visible surface).
These cases are represented in the following figures that highlight the profiles of the two surface parts: SC is in blue, SV in red.

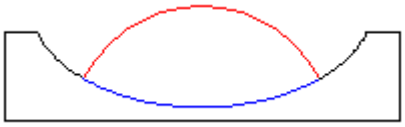
(fig.10)



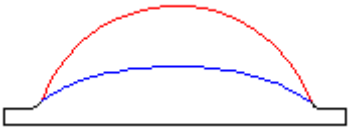
(fig.11)



(fig.12)

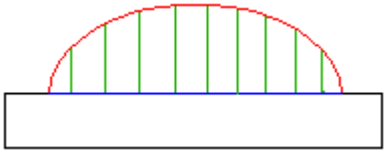


(fig.13)

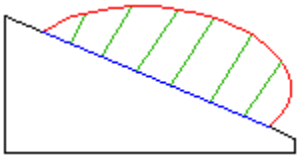


One of the key principles underlying the fluid simulation is based on this pre-emptive splitting of the outer surface.
Indeed, considering the two surfaces, it is possible to establish a relation that starting from SC enables the very rapid calculation of SV's shape.
Let us see how by examining a new version of the previous graphs.

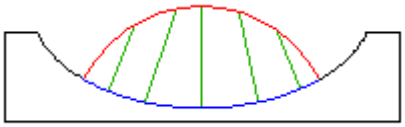
(fig.14)



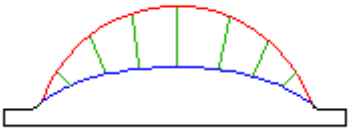
(fig.15)



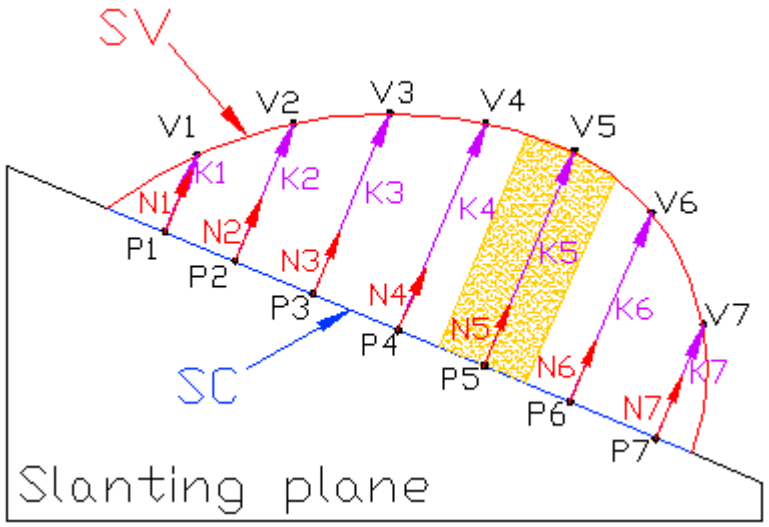
(fig.16)



(fig.17)



These figures differ from the previous ones due to the presence of the green segments that link some points of SC to SV.
As you can see, these segments are perpendicular to SC, and actually represent on SV the weighted projections normal to the plane that SC lies on in a number of preset points.
The concept can be analysed in greater detail in diagram (18).



(diag.18)

The points that have been called $P_{1...7}$ are the control points on SC, and vectors $\vec{K}_{1...7}$ are the extensions of SC's (weighted) normal $\vec{N}_{1...7}$ leading to the above points.

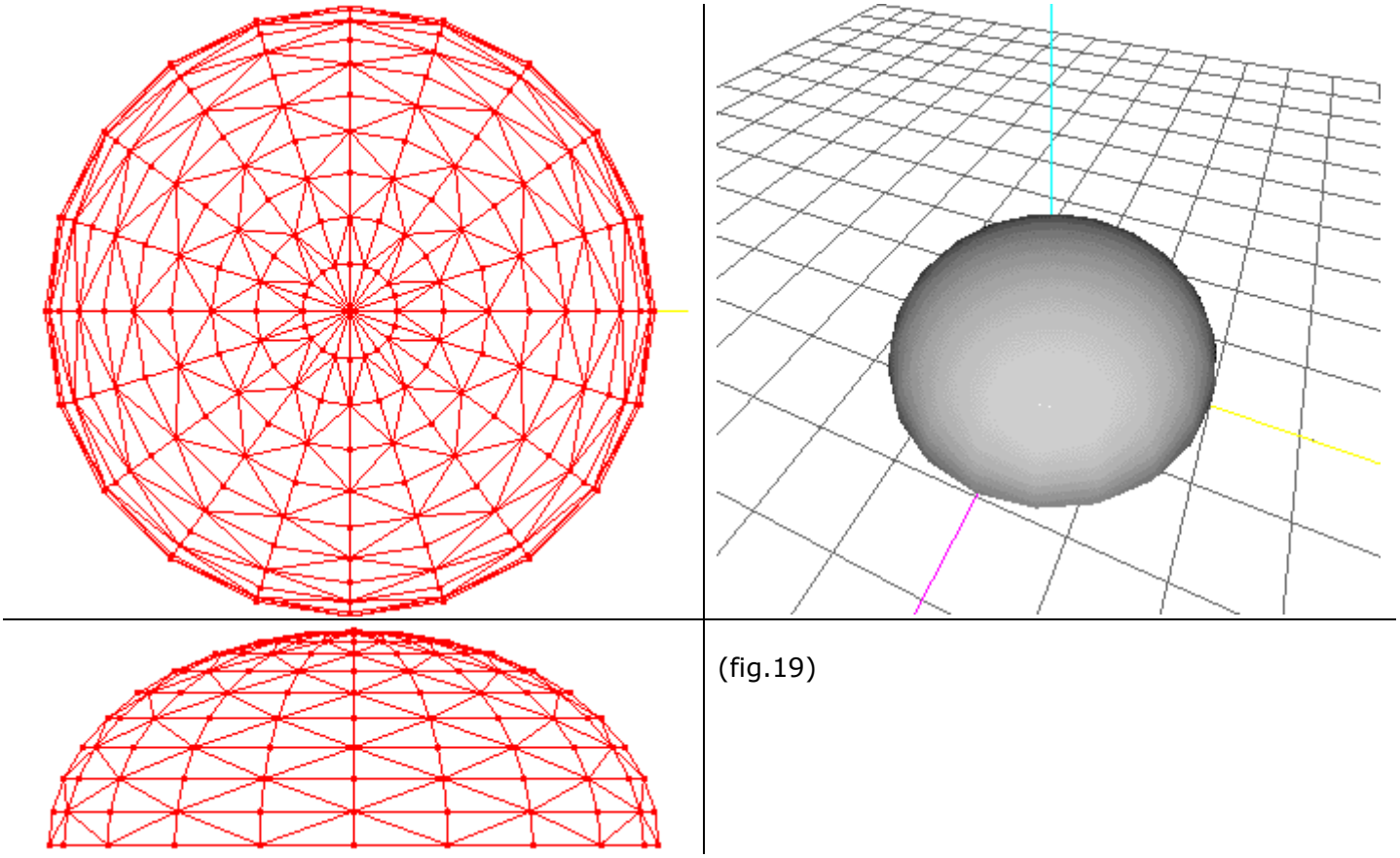
The vector's magnitude determines the location of the $\vec{K}_{1...7}$ points that outline (discretizing it) the SV surface.
It is determined based on the quantity of fluid that is expected to fit in the cylindrical neighbourhood between each P and its related V.
The section of a hypothetical neighbourhood is represented in the figures by the yellow section cross cut by $\vec{K}_5$.

The control points move on the slip surface in reference to its local inclination, the viscosity of the fluid and the repulsive force existing between the control points, that are simulated to force a distribution on the points in order to comply with restriction **d)**.
The following paragraphs will show how the above calculations will be managed.

11. Setting up the model

Once a "typical" behavioural model has been set for the fluid, the analysis can be extended to more precise shapes that can exactly describe how each vector $\vec{K}$ has to be managed in view of the different structural parameters.
As we will see, considering an initial geometrical model complying with items **a)** and **c)**, a set of relations can be created to also satisfy items **d)** and **e)**, using the concepts that were analysed above.

Reversing the problem, and starting with an initial situation where SV is known and hence also the set of V points, it becomes clear that the set of P control points pertaining to SC can also be established.
The most favourable condition is that in which the considered mass leans on a horizontal plane and is subject to the f.o.g., as shown in figure (19).



(fig.19)

This SV surface fully meets items **a)** and **c)** when at rest.
If we were to project the V points perpendicular to the horizontal support plane we can achieve the initially requested set of P points, that in turn govern the V points during the simulation phase.

This "inverted" construction phase can be considered as being the initial setup of the geometry of the employed fluid.

This shows that by calculating the vectors $\vec{K}$ during the initial phase we can also fully meet conditions **d)** ed **e)**.

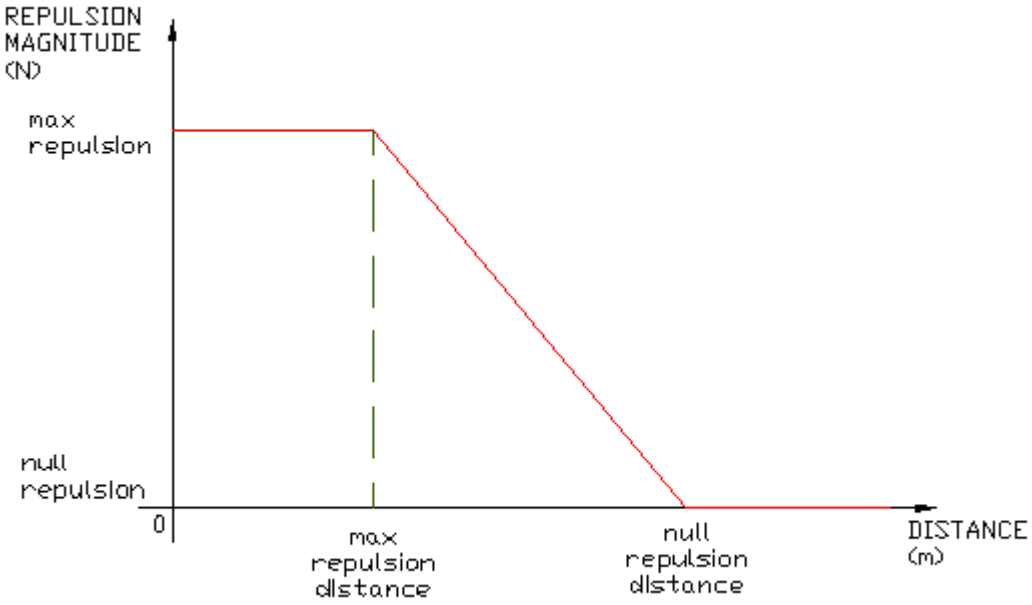
12. Animating the control points

Before proceeding to calculating the points of SV it is essential to understand how the control points of SC behave.
The principles analysed in the paragraphs on surface tension, internal friction and basic dynamics, have contributed to our understanding of the behaviour of the fluids considered in the case studies.
Now these cases and the principles of dynamics shall be taken as reference to describe the behaviour of the contact surface SC and the related control points P that outline it.
It is known that SC changes in function of the quantity and the distribution of matter in each point of the mass.
Hence we can imagine SC as being like a "fabric" that rests and adapts itself to the slip surface, responding to the latter's morphological features.
The behaviour of each point of SC shall therefore be restricted by the following factors.

- Force of gravity;
- The local gradient (detected in the neighbourhood of the control point);
- Adhesiveness to the surface (damping factor "B", analysed anon);
- Intermolecular forces simulated by means of reciprocal repulsion hindering the mass from occupying too small a surface, which might break the conservation condition for volume/mass;

It can be easily inferred that, to find a solution to this system of restrictions, it is necessary to set some rules that enable the program to recalculate the right location of each point respecting the aforementioned restrictions for each frame.
The f.o.g. is introduced into our system as a vector perpendicular to plane XZ and oriented "downwards" (Y-) with a magnitude of 9.81, namely the value of $\vec{g}$ expressed in m/s^2.
The local gradient, on the contrary, is represented by a unit vector (a vector whose length is one unit) that shows the direction in which the same point must move compared to the location of point P.
Adhesiveness to the surface is considered as being a "corrective" factor of the magnitude of the point's direction, namely a scalar used to simulate a force that hinders or restricts the mass while sliding.
This concept will be introduced anon, and will be called factor "B".
The intermolecular forces are introduced to enforce a more uniform as possible distribution of the control points, thereby avoiding the presence of accumulation points and conditions that could violate the principles inherent to the conservation of the mass and of the volume and illustrated in the introduction.
The forces will only include a repulsive component as the attractive component will be implicitly represented by the model through the preservation of the mass's structural integrity, that may be deformed, lengthened or extended but will never break or separate into several unconnected components.
Hence, an arbitrary point P_a shall have a representative mass value (that shall also be useful for the calibration of the model's behaviour) and will exert a repulsive force on all the other points the magnitude of which will be calculated in function of the distance separating P_a from the others.
The magnitude of this force may be changed with a function of choice based on the specific behaviour we want to assign the model.
For the sake of simplification and efficiency, in this specific implementation we shall use a repulsive reaction that is particularly suited to simulating muddy masses.

Diagram (20) represents the specific repulsive model created to this end.
The abscissa axis represents the distance between the two hypothetical P_a and P_b points whereas the ordinate represents the magnitude of the repulsive force between the two.



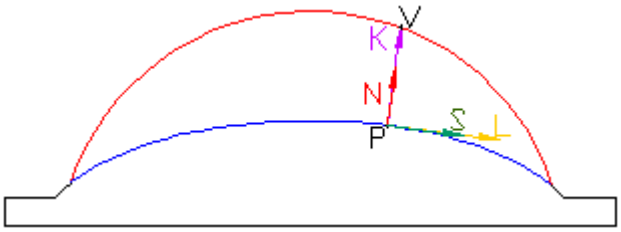
(diag.20)

The maximum and the minimum (null) repulsion distances are calculated by simply analysing the structure and the distribution of the control points in the initial setup condition.
Knowing that the distance amongst the control points must be as uniform as possible, it can all be simplified resorting to an easy and effective solution.
Considering the set of SC points taken in separate sets of two, the minimum detected reciprocal distance can be established which, in our model, represents the maximum repulsion distance.
Based on this same concept, it is possible to imagine the null repulsion distance as being the average detected reciprocal distance.
As can easily be expected, (and assessed by means of the demonstration program), the model will thus tend to flatten the mass by extending the SC surface once it has contacted the support surface (unless the latter is very concave), thereby simulating the adaptation and redistribution of the mass to suit.
The maximum repulsion is a parameter that can be set to suit, based on the specific behaviour we want to achieve from the simulation.

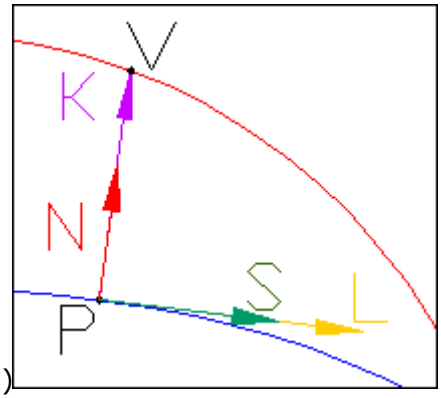
13. Basic form of FRF algorithm for calculating SV :

In order to meet both **d)** and **e)** a method needs to be formulated to correctly calculated both the length and orientation of vectors $\vec{K}$.
Considering diagrams 21 and 22 (that take into detailed consideration only one control point of SC), it is possible to describe a number of essential geometrical shapes.

(fig.21)



→



(fig.22)

Let us consider the following factors:
P → Control point located on SC
$\vec{N}$ → Normal at the slip surface on which the related point P is located
$\vec{K}$ → Weighted extension of the point P's normal $\vec{N}$, calculated in function of the expected mass of fluid included in the circular neighbourhood between P and its related V
V → Point located on the SV surface corresponding to P and calculated based on the displacement represented by $\vec{K}$.
$\vec{S}$ → Unit vector indicating the direction that P moves in due to its sliding on the support surface.
$\vec{L}$ → Vector representing an acceleration whereby P changes its motion sliding on the support surface.
$\vec{H}$ → Vector indicating the P's displacement in the time unit.

Hence, to calculate the shape of the SV surface, it will be sufficient to calculate the new position of the V points for each frame, updating the position of the control points P and related vectors $\vec{N}$, $\vec{K}$, $\vec{S}$, $\vec{L}$, $\vec{H}$.

The ***FRF algorithm*** needed to calculate all the V points will hence be made up of all the basic steps described as follows.
For each P point:

1. Obtain the heights of the slip surface in the neighbourhood of P to determine the sliding direction $\vec{S}$
2. Calculate $\vec{S}$ starting from the data gathered on P's neighbourhood;
3. Calculate the resultant $\vec{F}_p$ of the repulsive forces existing between P and all the other control points of SC
4. Calculate $\vec{L}$ using the law of uniformly accelerating motion on an inclined plane;
5. Adjust $\vec{L}$ in function of the repulsive forces in place between the control points of SC
6. Calculate the step $\vec{H}$ performed in the time interval Δt, that is the duration of the previous frame
7. Adjust step $\vec{H}$ in function of the damping factor, introduced to simulate the friction between SC and the slip surface
8. Calculate the new position of P
9. Calculate normal $\vec{N}$
10. Calculate the vector $\vec{K}$
11. Calculate V

14. FRF Algorithm implementation details

We shall now analyse the specific implementations of the steps of the aforementioned algorithm.

1. The heights of P's neighbourhood are obtained very simply by performing a scan of the model of the support surface, gathering a preset number of points at a distance close to P at regular angular steps.

2. Vector $\vec{S}$ is calculated choosing the lowest point ($\vec{P}_{\min}$) detected by the scan in the circular neighbourhood of P and establishing a vector represented by the equation
   $\vec{S} = \vec{P}_{\min} - \vec{P}$

3. Considering the set of control points, the repulsive force $\vec{F}_p$ exerted on a specific P is provided by the sum of the vectors-force $\vec{F}_p$ of the repulsion of each point in the set on P itself.
   This relation is described by the equation
   $$\vec{F}_p = \sum_{i=0}^{n} \vec{F}_i$$
   As was previously mentioned, each $\vec{F}_i$ depends on the distance of the P point in question from all other points in the set.

4. $\vec{L}$ shows the displacement acceleration of P on the slide plane, and is calculated in function of the local inclination of the same flush against the position in which P is located. Considering vector $\vec{C}$ representing the f.o.g., vector $\vec{L}$ is calculated by projecting $\vec{C}$ on the unit vector $\vec{S}$.
   The projection is calculated using the formula
   $$\vec{L} = \vec{S} \cdot \frac{\vec{C} \cdot \vec{S}}{\left\| \vec{S} \right\|^2}$$

5. $\vec{L}$ is adjusted summing the acceleration that the P point is subject to in function of $\vec{F}_p$ and of its own mass m alone.
   $$\vec{L} = \vec{L} + \frac{\vec{F}_p}{m}$$

6. $\vec{H}$ is calculated by establishing the space covered by P solely in function of the acceleration $\vec{L}$ in time interval Δt, namely the duration of the previous frame taken as reference as time quantization.
   $\vec{H} = \vec{L} * \Delta t$

7. $\vec{H}$ is adjusted by multiplying it times a damping factor "B" defined in range [0,1], that shows the adhesiveness of P (hence of SC) to the slip surface.
   With B=0, there is a full damping effect; the mass adheres to the slip surface hindering any displacement.
   If B=1, the damping is ineffective and the mass slides as though there were no friction or adhesiveness between the mass and the slip surface.
   If 0<B<1, the behaviour of the mass is proportional to B, simulating a greater or lesser adhesiveness to the slip surface.

   $\vec{H} = \vec{H} * B$ with $0 \le B \le 1$
   (warning: this method does not take into account Δt time quantization!).

8. P is updated by simply considering it as a vector that identifies the location in space and summing to it $\vec{H}$ applied to the time interval Δt, namely the duration of the previous frame.

   $P \leftrightarrow \vec{P}$

   $\vec{P} = \vec{P} + \vec{H} * \Delta t$

9. Normal $\vec{N}$ is achieved by considering P and any two points in height, A and B, identified in the neighbourhood of P in point 1. By considering P, A and B as vectors, two support vectors are achieved as follows.

   $P \leftrightarrow \vec{P}, A \leftrightarrow \vec{A}, B \leftrightarrow \vec{B}$

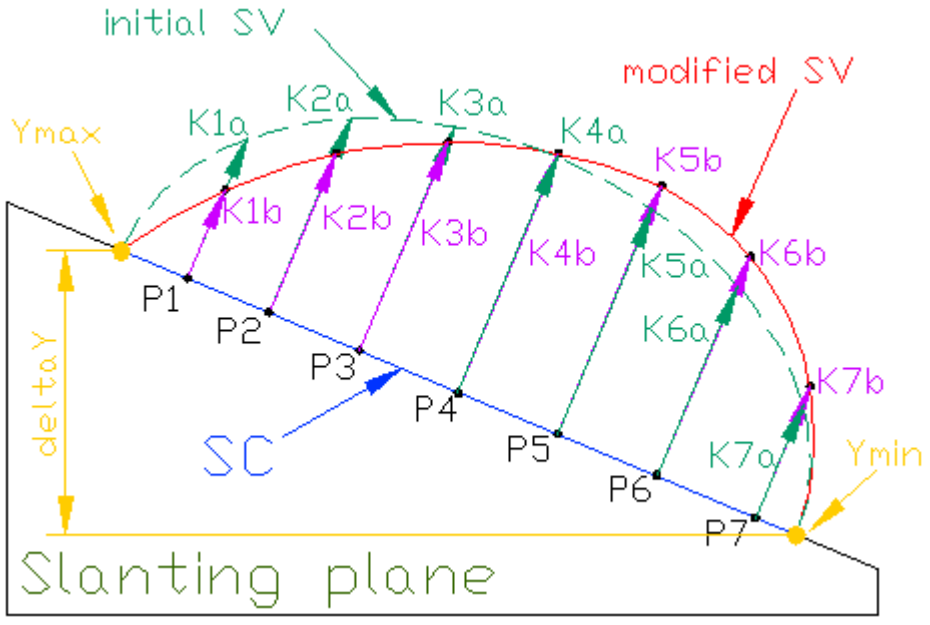
   $\vec{I} = \vec{A} - \vec{P}$

   $\vec{J} = \vec{B} - \vec{P}$

   Vectors $\vec{I}$ and $\vec{J}$ can be used to calculate $\vec{N}$ as their cross-product (represented by the symbol "*")

   $\vec{N} = \vec{A} * \vec{B}$

10. As anticipated, vector $\vec{K}$ is obtained creating a weighted extension of $\vec{N}$. The principle underlying the weighting is quite simple. Above it was explained that the magnitude of $\vec{K}$ is equal to the distance between V and its related P, hence representing the quantity of mass in the cylinder neighbourhood the axis of which is $\vec{K}$. It becomes intuitively clear that we need to establish what *percentage of the mass* is present in this section of the body compared to the initial stable condition where the fluid was shaped like a "meatball". As the difference in height Δy between the lowest point $Y_{\min}$ and the highest point $Y_{\max}$ of SC is known, we can estimate a *plausible distribution* of the fluid's mass, taking into account that the latter will tend to shift from the higher points to the lower ones accumulating there due to the f.o.g. (the concept can be better understood by referring to figure 18 above). By considering the "*percentage mass*" as a variation of the mass against its start condition and hence as a percentage variation of the original distance $\overline{PV}$ (that from now on will, for the sake of simplicity and form, be called $\overline{CV}$, where C represents P in the initial position), it becomes much simpler to devise a solution. The employed method can be observed in figure 23.



(fig.23)

The factors involved in the diagram include:

$\Delta y = Y_{\max} - Y_{\min}$

$P_y =$ value of the height of the examined P point.

The green dashed line represents the profile of the mass at rest.
Vectors $\vec{K}$ that discretize the surface are represented by the annotation $\vec{K}_{1...7}a$.
The red line represents the profile of the mass in a condition where it is reacting to the f.o.g and to the inclined slide plane.
The vectors $\vec{K}$ that discretize its surface are represented by the annotation $\vec{K}_{1...7}b$.
There is evidently quite a difference in magnitude between each $\vec{K}_i a$ and the related $\vec{K}_i b$ that will be calculated anon as a percentage difference.
Hence

$\vec{K}_i b = \mu_i * \vec{K}_i a$ (where μ_i is the scalar factor that changes the percentage of $\vec{K}_i a$)

whereby

$$\vec{K}_i b = \mu_i * \vec{N}_i a * \left\| \vec{K}_i a \right\| = \mu_i * \vec{N}_i a * \overline{CV}$$

where $\overline{CV}$ (as mentioned previously) is the distance $\overline{PV}$ in an initial stable condition, as mentioned above.
The calculation of μ_i requires some explanation as to the role held in its specific context of use.
As shown in the above formula, it represents a percentage adjustment of $\vec{K}_i a$.
Its value shall hence be included in a range $[\mu_{\min}, \mu_{\max}]$ of an amplitude $\Delta\mu$ of 100% (therefore 1 if expressed normalized to the unit).

Therefore, setting the restriction
$\Delta\mu = \mu_{\max} - \mu_{\min}$ =100% = 1
$\mu_{\min}$ and $\mu_{\max}$ can be assigned values at will, such as for example
$\mu_{\min} = 0.25$
$\mu_{\max} = 1.25$
or
$\mu_{\min} = 0.10$
$\mu_{\max} = 1.10$
depending on the shape that is to be applied to SV.

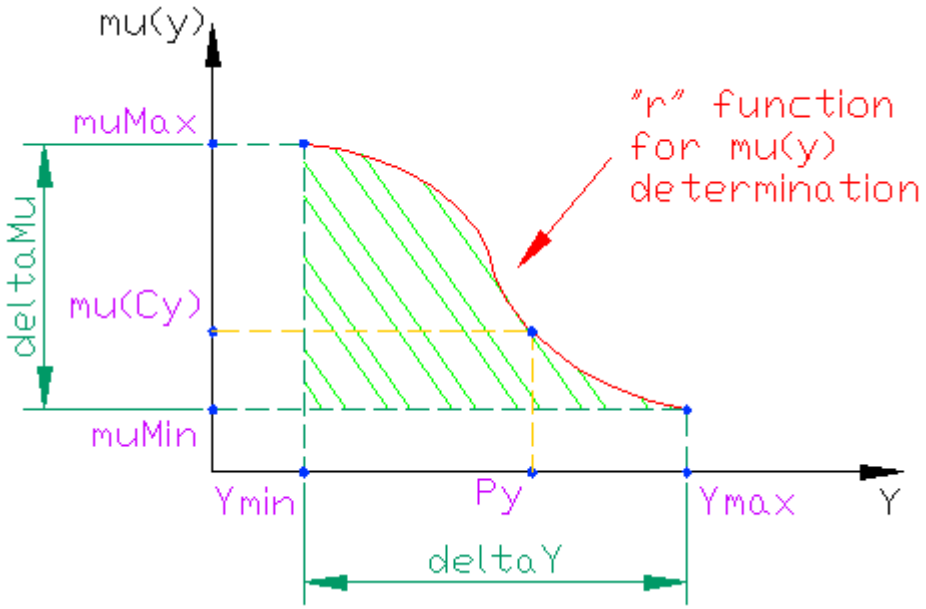
Clearly, this freedom in assigning the far ends of the range adds remarkable flexibility to our simulation model.
Given that, as has been mentioned, μ_i enables us to quantify the percentage difference of the local mass at a specific P point compared to the reference case, it is necessary to define μ_i as a variable that depends on Δy and P_y as follows.

$\mu_i = r(\Delta y, P_y)$

Where "r" is a function that has been included for the sake of simplicity in the formulation. Given that the aim is to simultaneously meet the conditions set by **a)**, **b)** e **c)**, we need to ensure that for each frame in which has been established the value of μ_i in function of P_y

represents a continuous and differentiable function, to avoid generating discontinuity in the model's behaviour.

Graph 24 shows a hypothetical case of a (sinusoidal) function "r" that fully meets the requirements.



(graph 24)

This function "r" is generated by the following relation that, as is clear to all, considers all the involved parameters providing a suitable sinusoidal interpolation that can satisfy the above stated requirements.

$$\mu_i = \mu_{\min} + \frac{\left(\Delta\mu * \left(\cos\left(\frac{(P_y - Y_{\min}) * \pi}{\Delta y}\right) + 1\right)\right)}{2}$$

whereby if

$$\vec{K}_i b = \mu_i * \vec{N}_i a * \left\|\vec{K}_i a\right\| = \mu_i * \vec{N}_i a * \overline{CV}$$

by replacement we achieve

$$\vec{K}_i b = \vec{N}_i a * \left\|\vec{K}_i a\right\| * \left(\mu_{\min} + \frac{\left(\Delta\mu * \left(\cos\left(\frac{(P_y - Y_{\min}) * \pi}{\Delta y}\right) + 1\right)\right)}{2}\right)$$

hence

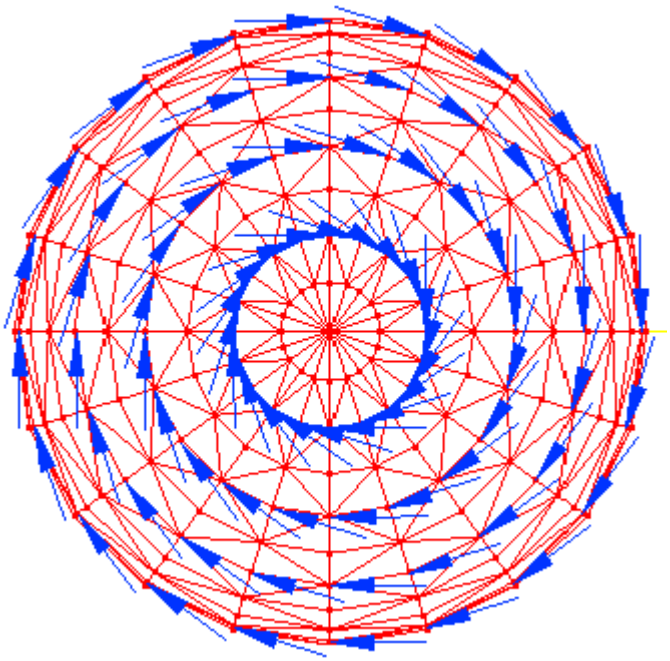
$$\vec{K}_i b = \vec{N}_i a * \overline{CV} * \left(\mu_{\min} + \frac{\left(\Delta\mu * \left(\cos\left(\frac{(P_y - Y_{\min}) * \pi}{\Delta y}\right) + 1\right)\right)}{2}\right)$$

11. Each V point is calculated considering the related P in a vector shape and adding to it $\vec{K}_i b$ obtained in the previous point. Hence the relation:

$$P \leftrightarrow \vec{P}$$
$$V \leftrightarrow \vec{V}$$
$$\vec{V}_i = \vec{C}_i + \vec{K}_i b$$

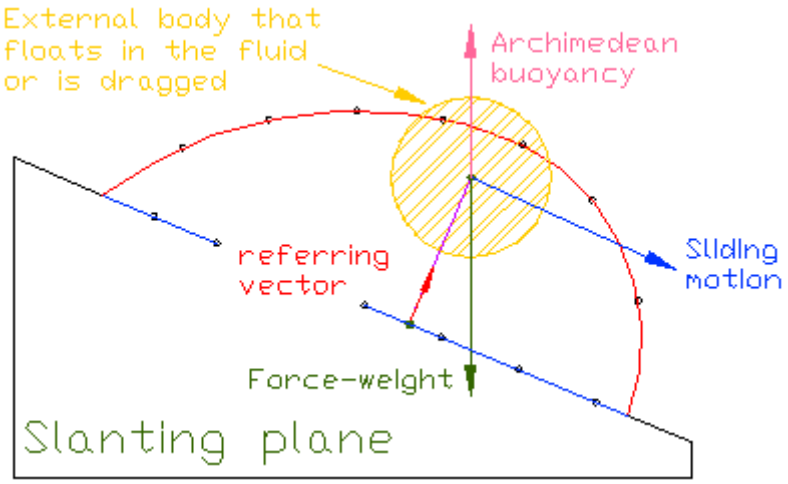
15. Conclusion of the model assessment and further suggestions for its application

The analysis of the simulation model based on the ***FRF algorithm*** ends here.
All the physical and mathematical principles needed for the implementation have been analysed and illustrated above and therefore all the issues that the simulation is based on have been clarified.
By considering the basic notions of fluid statics introduced in the first part of this document it is possible to understand the flexibility that this approach provides to the physical model.
Indeed, the simple modifications to the conditions of use makes it easy to simulate even complex and realistic behaviours.
Take for example the cases described in drawings **(3)**, **(4)** and **(5)**, where the features of the considered liquids responded to the interaction of both external and internal centrifugal forces, to pressure, to the Archimedean buoyancy, etc.
The aim of analysing all these cases has been to understand how easily they can be implemented by applying to the model the right system of forces that normally make real fluids behave as described.
Clearly, some require more modifications to the model than others, but generally the modular structure that the same projects has been implemented with provides a wide range of freedom.
In drawing (3) for example, the only body to be involved in the model is the same mass, hence a system of forces can be applied as illustrated anon in figure (25).



(fig.25)

In drawings **(4)** and **(5)** the mass interacts with external bodies: this interaction can be established by introducing additional calculations based on the vector scheme shown in figure (18), after having suitably analysed the vectors as shown in figure (26).

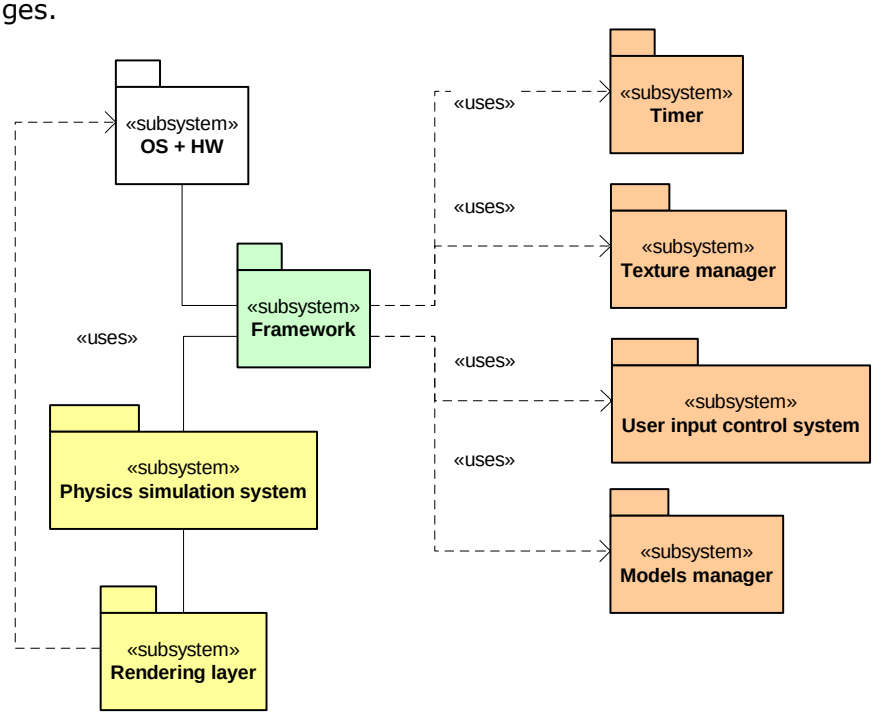


(fig.26)

Also bear in mind that the described model can be remarkably optimised in order to provide better performances.
This can be achieved for example by replacing the costly (in terms of machine cycles) trigonometric functions used in, with suitable look-up tables or approximation functions but these are implementation details that will not be addressed herein.

16. The project's basic structure

As stated in the introductory notes, this project will be made up of a modular and flexible structure that will enable not only future implementations and improvements but also extensions and changes.



(diag.27)

Diagram (27) shows the basic structure and links existing amongst the program's main elements and subsystems.
The employed graphical annotation is that defined by the specifications of UML (Unified Modeling Language).
Each represented functional block consists in a subsystem of the project having well-defined aims that are disjointed and decoupled with those of the other blocks.

The relations represented by the dashes and arrows are "uses" type relations and show (in the direction of the arrow) which systems are used by other different types of systems.
The relations represented by a single line without an arrow, show cooperation or an aggregation link.
Each subsystem has been associated to a colour in order to easily and rapidly identify the category they belong to.
The most important subsystem is the Framework (in green at the centre), that is made up of a set of structures and methods that can provide the simulation and rendering modules access to the services needed for their correct execution.
It also coordinates the system's start-up, refresh, high-level rendering and shut-down.
The services (subsystems on the right, in orange) each manage a specific resource, be it an input, a memory zone, a time domain etc.
The services are used by the physical simulation system by means of the Framework.
The simulation system (in yellow) holds the instances of the physical and geometrical models, in addition to the mathematical and physical functions needed to operate on the above.
The rendering layer (in yellow again) is in charge of the low-level processing of geometries, in this case represented by the API OpenGL.
It is directly "instructed" on how to perform by the physical simulation system that manipulates the employed geometries used in the three-dimensional representation of the scene.

It is clear that the employed project solutions enable:

- A high degree of orthogonality amongst objects, realised by means of the suitable implementation of the concepts of "high cohesion" and "low coupling".
- Good reflection ability, also known as "isomorphism"
- High code reuse (almost null redundancy in functional blocks)

To pursue these objectives, the whole project is based on a number of rather well-known design patterns that will be mentioned and analysed anon.
Just for information, remember that the most common definition of a design pattern states that it is a "a consolidated solution to a recurring problem in a specific environment."

17. System architecture and relations amongst classes

The system has been implemented maintaining a strong orientation towards objects, using creational patterns that make it extremely easy to renew the features and expand the architecture.

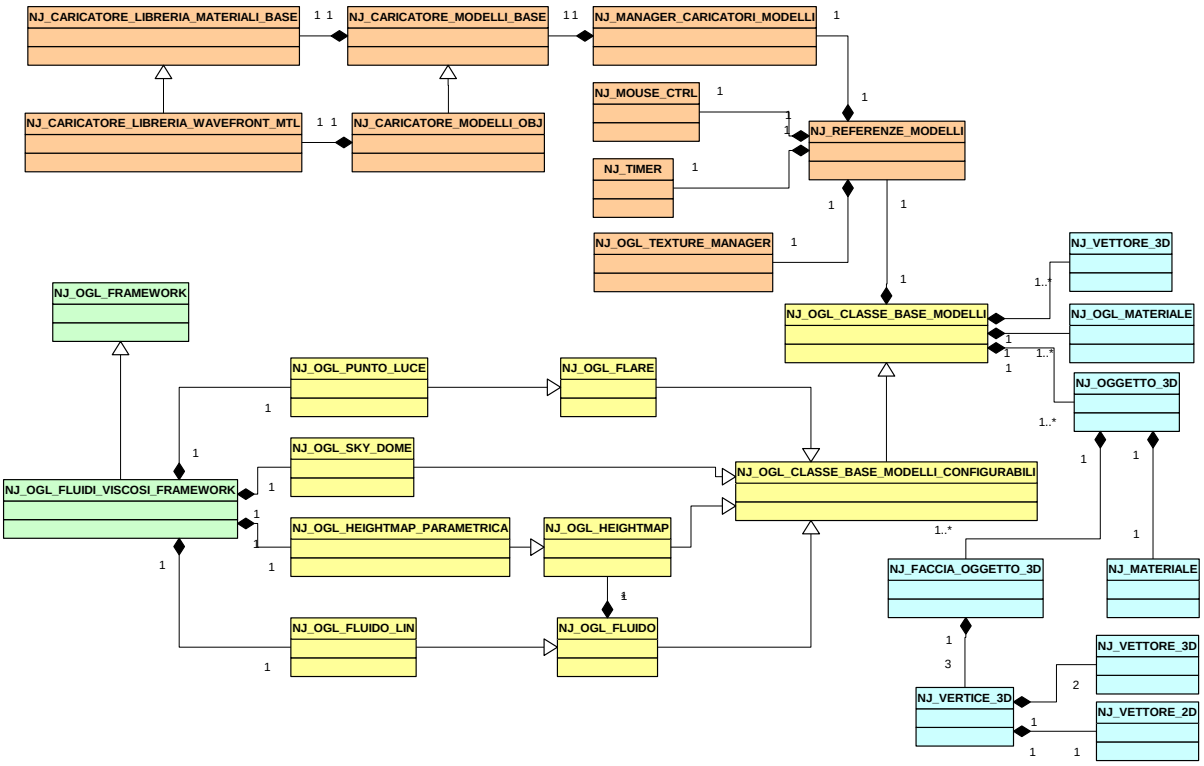
The use schemas and patterns that have more frequently been taken into account while designing and implementing the system's specific architecture include:

- Creational pattern "Builder"
- Creational pattern "Factory method"
- Structural pattern "Composite"
- Behavioural pattern "Mediator"

Some of the above have been applied in strict compliance with the specification whereas others have been suitably changed and adapted in order to more successfully fulfil their role.

Diagram 28 shows a global view of the relations amongst the basic classes implemented in the system and that make up the functional blocks described in diagram 27.

(diag.28)

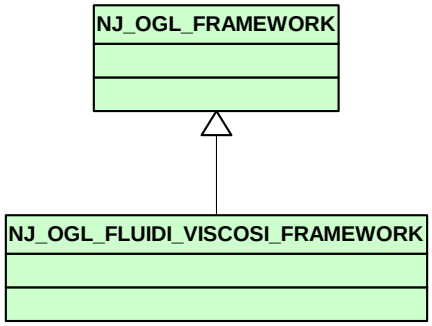


As in diagram 27, UML is used in the graphical notes.
The empty triangular arrows show a relation based on inheritance and the arrow indicates the reference class, as the represented concept is that of an "extension of".
Normally, in UML specifications, diamond-shaped arrows show a strong relationship based on aggregation, but to facilitate the understanding of the diagram, in this context they have also been used to indicate generic aggregation, instantiation, possession.
The arrow shows the "owner", "the referentiator" or "allocator" of the instance.
The numbers at the side of the far ends of the relations show the cardinality of the links, specifying the specific or possible number of objects involved in the relation.
The list and the type of members in each class are not shown for a question of space and readability.

The colour difference amongst the classes in the diagram enables the identification of the various sections of the architecture divided based on type and function, and also the various above-mentioned patterns.
What follows are the diagrams for all the different sections through detailed extracts taken from the global representation.

18. The program's foundation: the framework

The objects in green are the base of the project, the framework that manages high-level operations, such as the start-up, the handling of the messages, the allocation of services, etc.
As you can see, the class NJ_OGL_FLUIDI_VISCOSI_FRAMEWORK extends the reference class NJ_OGL_FRAMEWORK.
Hence the functional base (NJ_OGL_FRAMEWORK) can be established that can be specialised and extended to suit using the features required by the specific application project while keeping the same use and interface pattern with services and hardware resources.



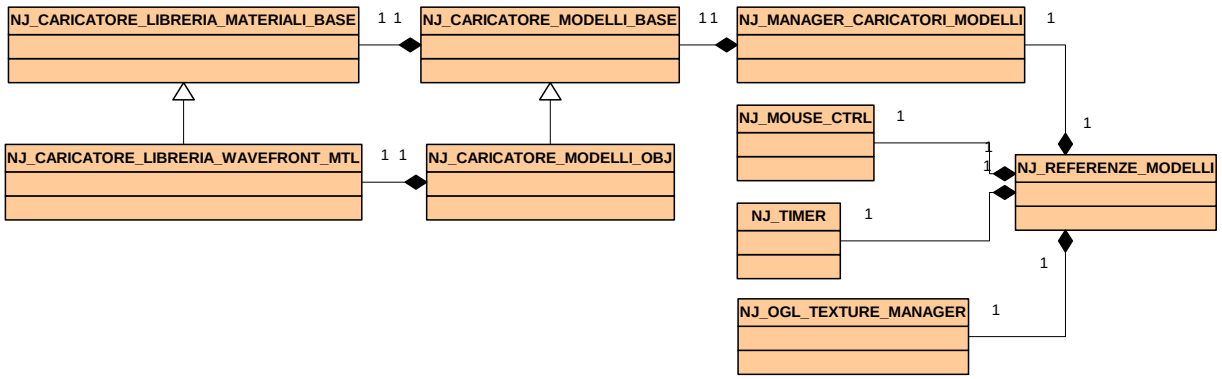
(diag.29)

19. Managing resources and interoperability of the service systems

The objects in orange are part of the section of the architecture in charge of providing interface and interoperability services to the simulation and rendering models.
What follows is a detailed description of this part of the architecture.
As you can see, the class NJ_REFERENZE_MODELLI brings together the instances of objects that are in charge of managing hardware and system resources.
These managers are:

- NJ_MANAGER_CARICATORE_MODELLI is in charge of interfacing the models (objects of 3D scenes) with the geometry loading functions. The latter are included and specialised in the classes NJ_CARICATORE_MODELLI_BASE and the extension thereof NJ_CARICATORE_MODELLI_OBJ, which in turn receive the materials library loading functions (textures, colours, light factors, etc) from NJ_CARICATORE_LIBRERIA_MATERIALI_BASE and the specific extension for the "wavefront obj" models called NJ_CARICATORE_LIBRERIA_WAVEFRONT_MTL.
- NJ_MOUSE_CTRL manages the interface of the framework (and related system) with the hardware resources associated to the mouse by means of Direct Input (DX). It enables the pointing device to be employed for the most disparate camera and scene control purposes.
- NJ_TIMER enables the timing of system events by means of an extremely precise performance counter: it is essential to synchronisation and time quantization of the scene and its related objects and movements.
- NJ_OGL_TEXTURE_MANAGER is in charge of controlling the generation, listing, allocation, association and use of the textures used by the scene's objects.

(diag.30)



20. Models and simulation components

The objects in yellow make up the real core of the system, that is the simulation and rendering components allocated by the framework and that obtain the required functions from the services by means of the framework.
As you can see, all the objects come from a reference class called NJ_OGL_CLASSE_BASE_MODELLI that outlines a precise functional structure in charge of the start-up, parameterization and rendering of geometries and the scene's objects.
It also contributes to formalise the main operational flows that make the object work correctly, namely:

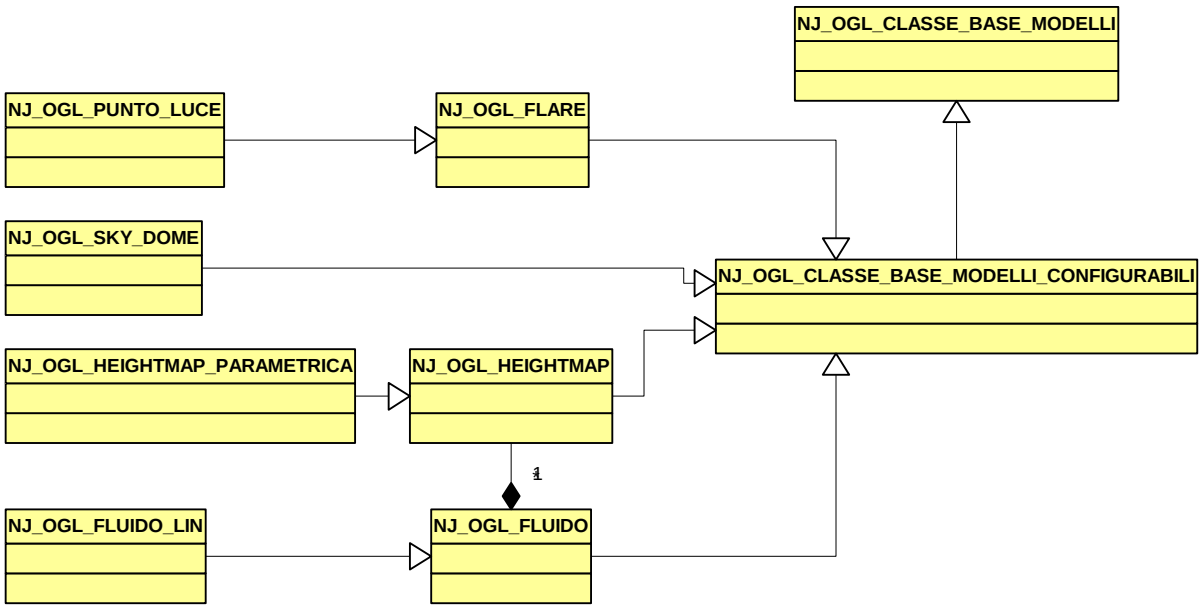
- Allocation of components
- Start-up and loading from file and/or from explicitly hard-coded parameters.
- Refresh synchronised with the system's timer
- Rendering
- Reverse allocation of components

A more modular solution might have consisted in dividing the coordination, refresh and rendering functions in 3 separate classes specifically dedicated to each purpose, in line with the MVC model (model, view, control).
This, however, would have increased the overall complexity of the system and prevent possible low-level optimisation operations.
It would also have implied the occurrence of drops in efficiency mainly due to the increase of the occupied memory by the virtual tables of the new classes and by the just-in-time type-cast at runtime amongst hierarchically different classes.
The chosen solution seemed to be the right compromise between efficiency and modularity.
The class NJ_OGL_CLASSE_BASE_MODELLI has therefore been used as a base for the daughter-class NJ_OGL_CLASSE_BASE_MODELLI_CONFIGURABILI, that extends the former's loading and parameterization features adding functions to the start-up flows.
These functions enable the parsing of specific configuration files (of the type "parameter=value") associated to files of the models of origin by means of the concatenation of the string bearing their name (name of configuration file = model name + ".info").
This enables to remarkably increase control on the objects loaded outside the programming environment, if required by the project's structure.
All the system's 3D models resort to these functions: indeed many of them were created by extending NJ_OGL_CLASSE_BASE_MODELLI_CONFIGURABILI.
An example is provided by NJ_OGL_FLARE, an object that creates a three-dimensional flare that simulates a light source of choice.
By extending this class, NJ_OGL_PUNTO_LUCE was created: in addition to creating an apparent source of light it really simulates it using GL_LIGHT from API OpenGL that interacts with the system's models illuminating them when the lighting is activated.
NJ_SKY_DOME simulates the sky dome and uses the texture specified by the model it is associated with during start-up.
NJ_OGL_HEIGHTMAP enables to define a geometry to which the terrain texture can be applied and provides some basic functions for managing collisions with the same.
NJ_OGL_HEIGHTMAP_PARAMETRICA extends NJ_OGL_HEIGHTMAP by adding the functions for loading the terrain surface starting from a specific trigonometrical formula of which one case define the amplitude, phase and frequency of both axes of the XZ horizontal plane.
NJ_OGL_FLUIDO is undoubtedly the most complex and important class in the whole system, as it manages all the operations described in the chapters on managing a fluid model.
It performs the following operations:

- Loads a geometrical model from file, including information on texturing and the applied material;
- Acquires morphological information on the model (that is expected to be shaped as an elliptic cap as per the specifications), defining the functional parameters of SV and generating the related SC.
- Calculates SC and SV for each frame, correctly refreshing the geometrical structure as described in the previous chapters
- Renders the geometry.

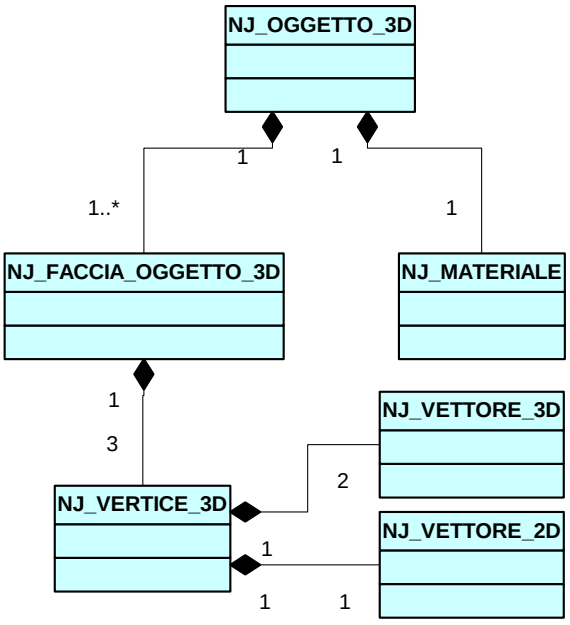
The class NJ_OGL_FLUIDO_LIN is an extension of NJ_OGL_FLUIDO that is in charge of simulating a model with a specific repulsive function in reference to the control points as described in figure 20 above.
By creating new extensions, new models can be achieved with different repulsive functions with the aim of achieving disparate behaviours of the fluid.
The decision to perform a separate implementation at different levels of the inheritance has the aim of recycling the largest number possible of control and code structures of NJ_OGL_FLUIDO while maintaining high developmental flexibility.
It is interfaced with NJ_OGL_HEIGHTMAP (through a composition link, instance acquisition) to achieve the necessary information on collisions and ground inclinations, thus correctly managing the sliding mass.
The experience gained with design patterns suggests that this interface solution between NJ_OGL_FLUIDO and NJ_OGL_HEIGHTMAP is not ideal as it does not meet the low linkage restriction. However, in this introductory paper, it was deemed unsuitable to increase the system's complexity by applying an in any case more effective interfacing or bridging model that could have enabled a more modular implementation of this relation.

(diag.31)



21. Data structures

The objects in blue are the data structures that are used by the system's various models to store the image of the state of the system and to perform simulation and rendering calculations.
As you can see, NJ_OGGETTO_3D is the most complex amongst the structures shown in the figure: indeed it holds the most important role in the game.
It acts as cornerstone for the scene's classes of models as it includes the information on the polygonal shapes to be rendered and the materials used in the process.
The polygonal shapes are stored in NJ_FACCIA_OGGETTO_3D as triangles, objects made up of 3 three-dimensional vertexes defined by the structure NJ_VERTICE_3D.
As you can verify in the source code annexed to this document, this structure includes both the point's position and three-dimensional direction in the shape of a vector (NJ_vector_3D) and the texturing coordinate (NJ_vector_2D).
As was mentioned above, the material is included in a dedicated data structure called NJ_MATERIALE that is loaded by the classes in charge of managing material libraries in the system.



(diag.32)

22. Performance and details

The program was completed and tested on a thus configured PC:
CPU: Athlon XP 2200+
Ram: 1GB CAS 2
Video Display Board: Matrox Parhelia 128
O.S.: Windows® 2000
DirectX: version 9.1
OpenGL: version 1.1

Using the specified hardware/software configuration and the geometrical model used for the fluid mass in the file "Data/fluido.obj" the following average performance was achieved in windowed mode at 800x600x32bpp an 32 bits zbuffer:

- Compilation in debug mode: 35 FPS
- Compilation in release mode: 110 FPS

A better performance can be achieved by reducing the number of vertexes on the test model presented above, as the computational load of the simulation procedures alone can roughly be calculated by squaring the number of vertexes in the model.

A number of both high- and low-levels of optimisation can be applied to the sources, but were not implemented in this release for the sake of greater readability.
The object of the scene's geometrical models were included in the following files:

- Data/fluido.obj -> initial geometry of the fluid mass
- Data/skydome.obj -> skydome of the three-dimensional simulation environment
- Data/terreno.obj -> ground heightmap

The behaviour of the simulation can largely be parameterized by adjusting the values specified in the configuration files associated to the various objects in the system.
In particular, reference can be made to:

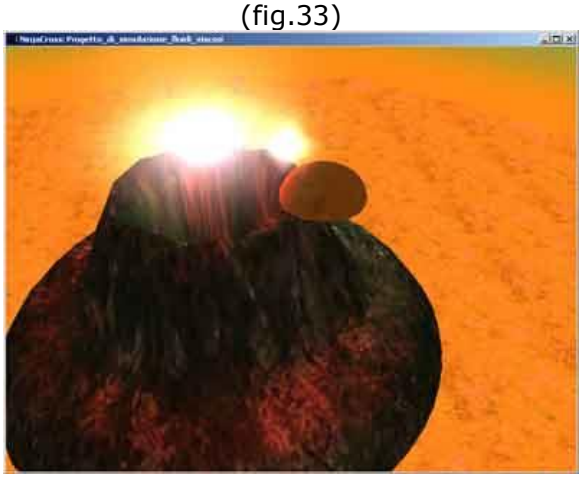
- Data/config.txt -> framework configuration file
- Data/fluido.obj.info -> configuration file of the fluid mass
- Data/terreno.obj.info -> configuration file for the parameterizable height map

It is possible, if you wish to monitor the execution of the program and of the various calculation and rendering routines, to resort to the NJ_DEBUG_LOG class present in the code.
The latter lists the suitable tracing methods that can be included at will in the sources and generates a log file in XML format called "debug.xml".
This file can also be referred to in order to research and solve errors if you want to experiment with the source code.
For the sake of simplicity, this project is available in Visual Studio 6.0 and Visual Studio.NET versions.

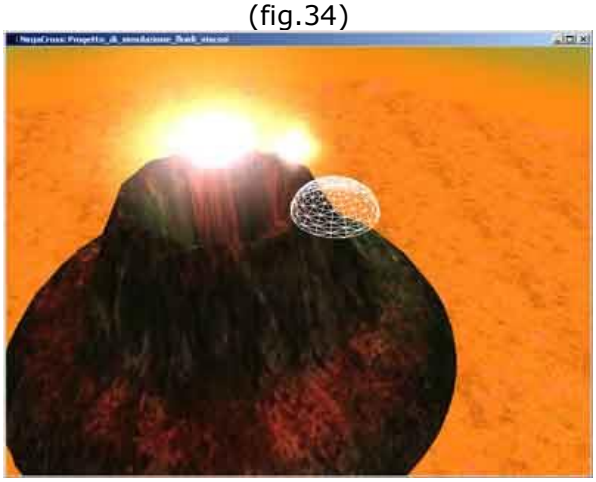
23. Program execution and output

Once the compilation has been finalised (preferably in *release mode* considering the excellent performance compared to that achieved in *debug mode*), the program linked to this document will represent a scene that represents the real-time simulation of a mass of lava that impacts and slides down the side of a volcano with the subsequent deformation.
The rendered mass will experience a first stage in flight during which it will maintain its shape (as the friction with the air is unknown).
The situation observed through the camera will be similar to that shown in the screen shots of figures 33 and 34.

(fig.33)



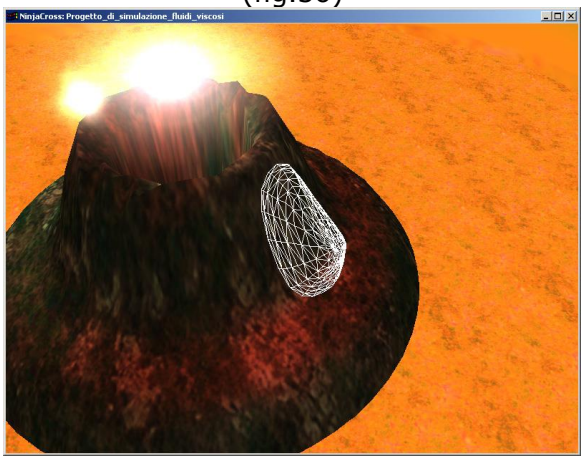
(fig.34)



Afterwards, due to the f.o.g., the mass will impact the ground and will be deformed due to the gradient of the mountain side.
After contact with the ground, all the control points in SC will start to slide downwards, hence changing the shape of SV.
This event can clearly be seen in the screen shots in figures 35 and 36.

(fig.35)

(fig.36)



Having reached the lowest and flattest point of the incline, the motion of the mass will tend to run out as the control points no longer receive any force that pushes them to move further forwards as the resultant of the vector of the f.o.g. and the sliding unit vector is almost inexistent.
Their residual kinetic energy will asymptotically tend to be zero due to the damping "B" introduced in the program that will cause the mass to continue with at a gradually lower speed.
Figure 37 shows this condition: it clearly shows the mass's evident change in shape due to the sliding buoyancy running out.

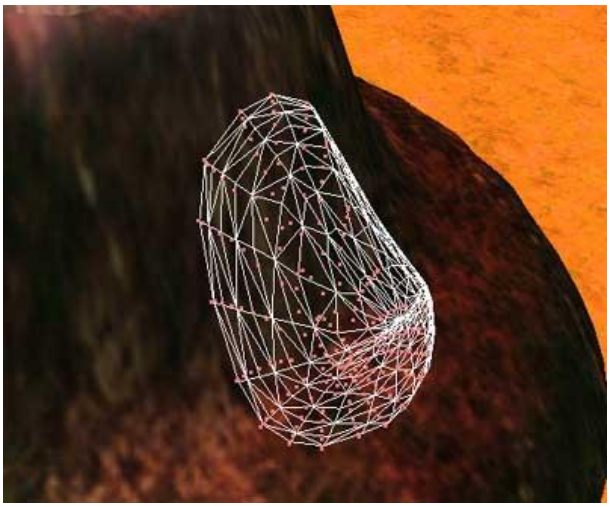
(fig.37)

24. Using the program

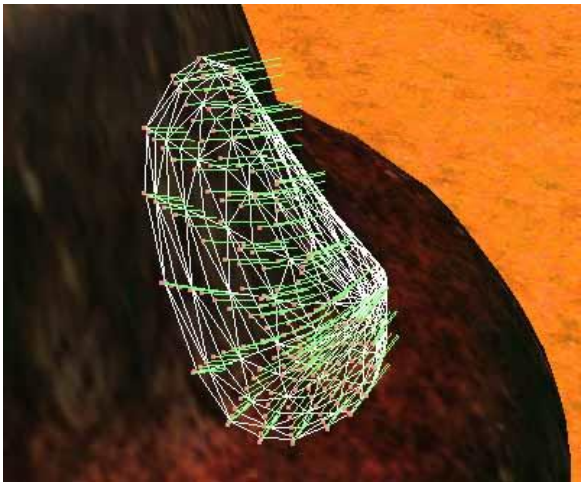
At run-time, the framework acquires the information from the mouse and the keyboard enabling simple and immediate navigation in the scene.
The mouse determines the camera's direction of observation and displacement, whereas the arrow keys define the shift in the four directions in line with the orientation of the camera.

The "Q", "A" and "Z" keys enable the mass rendering mode to be reset as regards the texturing mode, wireframe, vertex mode (as we say in figures from 33 to 36).

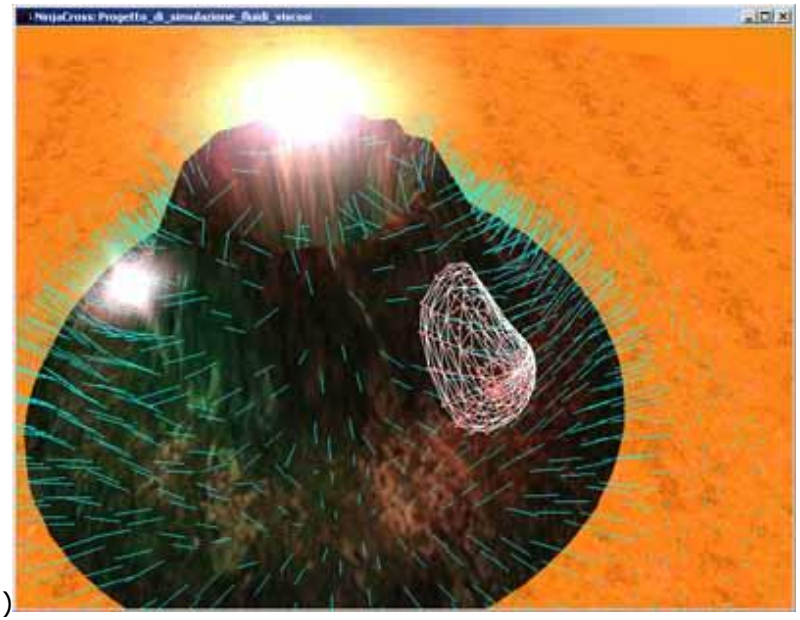
Keys "W" and "S" enable to activate or deactivate the rendering of the control points in SC and shown in red and only visible in a rendering mode other than the texturing mode.
Figure 38 shows the control points of SC rendered below the geometry of SV (in the white wireframe).

(fig.38) 

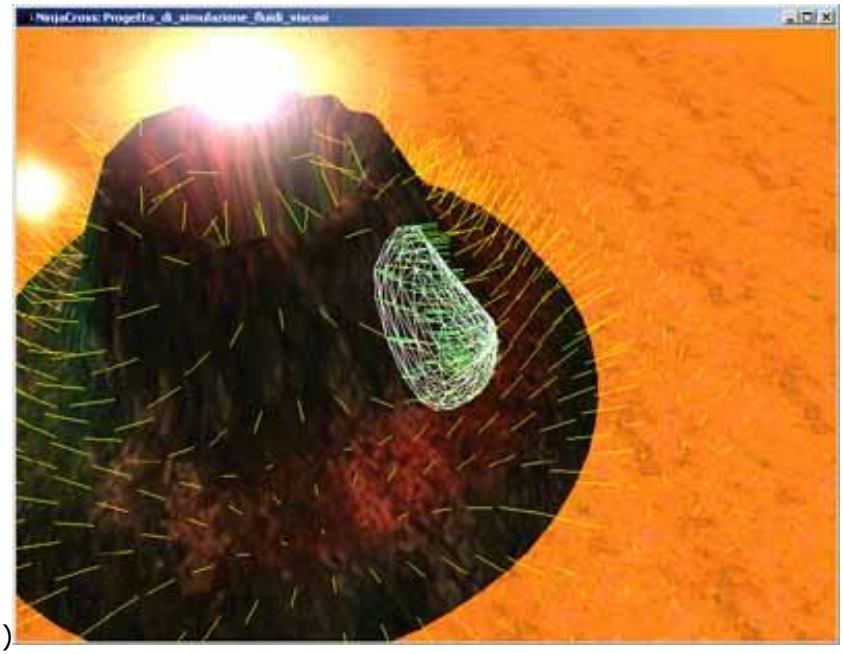
Keys "E" and "D" activate and deactivate the rendering of the normals on the surface of SC flush against all the control points.
Figure 39 shows this viewing case, which is useful to check that the internal algorithms of the object NJ_OGL_FLUIDO are functioning correctly.

(fig.39) 

Keys "T" and "G" activate and deactivate the rendering of the normals of the polygonal shapes of the volcano's surface (segments in blue), so as to provide visual proof of the consistency between the normals of SC and those on the slip surface.

(fig.40) 

Keys "R" and "F" activate and deactivate the rendering (segments in yellow) of the normals of the vertexes of the volcano's surface.
As for the normals of the faces, this visual rendering is can also be very useful to check the system's operating conditions.

(fig.41) 

25. Reference and bibliography

[1] Macrofisica e Microfisica – Paride Nobel (Editrice Ferraro)
[2] Fisica generale e sperimentale – Guido Piragino, Gualtiero Pisent
[3] Chimica – A. Post Baracchi, A Tagliabue
[4] Official web site of OpenGL www.opengl.org
[5] The Red Book, OpenGL Programming Guide - Addison-Wesley Publishing Company
[6] TheOpenGL Graphic System version 1.2.1 - Mark Seagal, Kurt Akeley
[7] Graphics Gems 5 – Alan W. Paeth
[8] Thinking in Patterns – Bruce Eckel
[9] Patterns and Software Development - Kent Beck

For any further information, request, or to send in your remarks, you can contact the author at the e-mail address info@ninjacross.com

Other tutorials and resources can be found on the web site http://www.ninjacross.com

Federico Coletto, 07/10/03