How Does Phoenix's Simulation Work?

This page contains information on how Phoenix simulations work internally. Steps for setting up a simulation can be found on the QuickStart Guides page.

How does Phoenix's fluid solver work?

Phoenix's fluid simulation calculates how a fluid would evolve during a period of time. While you might be familiar with the term *fluid* as meaning "liquid", in physics the term *fluid* refers to both liquids and gases. When this documentation refers to a *fluid*, it means liquid or fire/smoke. The simulation runs in sequential steps, and at each step the fluid is calculated a little further in time ahead of the previous step. This way each new step depends on the previous step, and a new step cannot start before the last one has already finished. This is one of the biggest differences between distributed rendering and distributed simulations - e.g. if you want to simulate 100 frames, you can not run frames 1-50 on one machine and frames 51-100 on another machine simultaneously.

The result of each simulation step is a velocity field - i.e. the fluid simulation calculates the speed and direction of the movement of the fluid in different points in space. In order to visualize the effect, a simulation can also transport visible fields, such as smoke density, temperature, and also different kinds of particles along the velocity field that gets simulated. The two most important processes in a simulation are **Advection** - how the fluid moves through space, and pressure solve, called **Conservation** in Phoenix - which is basically the step that makes simulated fluids behave like real-world fluids - swirling, rolling and creating plumes. Both of these options are controlled from the Dynamics rollout of the Phoenix Simulators.

A fluid simulation can also transport visible fields along the simulated velocity field. Such fields can be smoke density, temperature, different types of particles, etc. The velocity field is the raw result of the fluid simulation and by default it's invisible, but these additional channels make the fluid motion visible. You can enable these fields for simulation from the Output rollout and then actually create them in the simulation container using Sources, or other more advanced methods.

Each simulation step roughly does the following:

- Reads the scene nodes which interact with the simulator - obstacle geometries, emitters of fluid which can be either geometries or particles, forces, etc.
- Adjusts the simulation velocity during the **Conservation** phase.
- Transports the fluid along the velocity field during the **Advection** phase.

The simulation runs from a certain start frame to an end frame on the timeline. Each frame, one or more simulation steps can be calculated. This is controlled by the **Steps Per Frame** (SPF) parameter under the Dynamics rollout. In case there are fast moving geometries in the scene, or the fluid itself has to move quickly, it's good to use more **Steps Per Frame**. This way the scene state could be read more often and also the Advection can work in shorter intervals - otherwise the fluid movement will break up and will not be smooth. However, using more steps will require more time for the simulation to calculate, e.g. using twice more steps will take twice more time to simulate.

There is also another downside to using more steps - each time the fluid is moved during the **Advection** phase, it gets blurred, which is a part of the fundamental fluid simulation algorithms unfortunately. This means that the more steps pass, the more detail would be lost and the Temperature or Smoke channels will lose their fine details over time. So if a simulation at 1 **Step Per Frame** gets significantly smoothed and blurred after 1000 frames, the same simulation would lose that much detail only after 50 frames if 20 **Steps Per Frame** are used. And not only the visible channels lose their detail the more steps pass - the velocity field loses detail as well. So if a smoke plume rolls energetically into a mushroom at the beginning of the simulation, it would gradually lose its momentum and its shape will dissolve later. This is why it's always best to use as few **Steps Per Frame** as possible and increase them only when needed. Also note that the **Time Scale** parameter has the same effect - running the simulation with **Time Scale** of 0.1 for 50 frames will lose as much detail as it would lose over 500 frames with **Time Scale** of 1.0. This is why Phoenix offers several different methods for retiming a simulation after it has run once at regular speed.

At the end of each frame Phoenix saves the fluid data up to this moment to a new cache file. Cache files are numbered with their frame numbers by default, which can be controlled by the Output rollout. Note that if you have many steps per frame, a cache file would be saved only once per each frame - e.g. if you have 5 Steps Per Frame, the simulation will run 5 steps and just then a new cache file would be saved. Cache files make it possible to quickly scroll through the timeline and preview the simulated cache sequence, and also to render it without the need to simulate each time you need to go to a specific frame. Starting a new simulation would overwrite the existing cache files.

Additionally, cache files can be used later by Phoenix to run a Resimulation. It can be used in a number of different ways, such as adding more detail or re-timing an already simulated cache sequence to a different playback speed, or for creating bullet-time effects. What is common between all different resimulation methods is that they all come down to using a sequence of already simulated caches (a *base simulation*) in order to enrich or modify it.

While the simulation runs, it's in the background, and the user interface of 3ds Max remains active. You can change many simulation parameters (except for a few initial core parameters) during the simulation and see how they affect it. Rendering is also enabled during simulation, so you don't have to wait until the end of the simulation to render images and see how the simulation looks and fits into the scene.

Particle and Grid-based Simulation

There are two main approaches used in simulation systems:

- In a particle simulation, particles move through space and each particle carries properties of the fluid (*particle channels*), such as Age, Size, Viscosity, RGB color. Particles interact with each other - attract and push each other apart, or exchange properties such as color and viscosity. Phoenix uses particles for simulations of liquid effects such as foam, splash or mist. The more particles there are in a simulation, and the more they need to interact with each other, the more time the simulation would take.
- In a grid simulation, the simulation container is divided into cells (voxels) that contain the fluid's *properties (grid channels)*. Examples of commonly used grid channels are Temperature and Velocity. The name *voxel* comes from an analogy with 2D images made of pixels, but in 3D space these have volume and are called voxels. The voxels are static pieces of space, and the fluid flows in and out of them, so with time the voxel's channels can change. Phoenix uses grid simulations for gaseous effects such as fire and smoke. Grid-based effects are contained within a rectangular grid, while particle-based effects have no such space constraint.

Particles tend to look and act like individual droplets (or, in the case of a close collection of particles, a glob of droplets), while grid-based effects can be very nebulous and atmospheric in appearance and can gradually fade and thin out. A major downside of grid simulations, however, is that while particles keep their properties throughout the simulation, a grid simulation could gradually fill up with smoke or liquid, or alternatively - matter could disappear from it with time. This can be avoided by using more **Steps Per Frame** or higher **Conservation Quality** which improve the accuracy of the simulation, but this way the simulation takes longer and it's still a major challenge to try and simulate liquids with a grid solver.

A third hybrid method between particle and grid simulations are the FLIP simulations, which Phoenix uses for liquids. FLIP simulations take the best from both worlds and produce much realistic liquid effects quickly. FLIP liquid simulations were first added in Phoenix 3.0. Before that, liquid were simulated using the grid solver. In Phoenix 3, you can still open and simulate older scenes saved from Phoenix 2 using the old grid liquid solver. Note that Phoenix uses pure particle simulation for the secondary particle systems such as Foam, Splash, WetMap and Mist as these are simpler and don't suffer from issues that liquids in pure particle or grid simulations have, such as having to maintain volume and not collapse on themselves if put in a large container.

Fire/Smoke vs. Liquid Simulation

Simulations performed by Phoenix can be roughly divided into two major categories:

- Fire and smoke - Gaseous effects like fire, smoke, and explosions. These simulations are grid-based. Such effects tend to be buoyant, meaning they are lighter than air and so tend to rise against gravity.
- Liquid - Pouring or flowing liquids, bodies of water such as lakes and oceans, and any simulation that requires foam or mist, such as beer, coffee, or even waterfalls. These simulations use both a grid and particles. Such effects tend to react to gravity by falling when not held in place by a container.

These categories are for convenience only, and are not rigid. Once you've created a few basic simulations and have become more familiar with Phoenix, you will have a better grasp of how the tools work, and when creating an effect that is not strictly fire or liquid you'll know how to represent it most efficiently. For example, embers or thin smoke might seem to fall into the "fire" category, but in practice these effects might be better served by particles and thus could use some of the tools designed for liquids.

Simulation and Rendering

The results from Phoenix simulations are saved to cache files. After that, Phoenix can read this cache data for quick viewport preview, or for rendering. Fire /smoke simulations usually produce grid voxels and can additionally produce particles that accompany and enrich the fire/smoke effect. Liquid simulations mainly produce particles and Phoenix can also automatically convert them to voxels when saving cache files during simulation.

The Simulator node can load cache data either from files simulated by Phoenix, or from other software. It can convert loaded grid voxel data to mesh, mostly needed for rendering liquid surfaces, though it's not a problem to mesh fire/smoke data as well. Phoenix meshes are like any other scene mesh geometry and you can apply standard render materials over them. Phoenix offers several surface render modes in the Simulator's Rendering rollout - **Mesh** , **Isosurface**, **Cap Mesh** and **Ocean Mesh**. While the mesh modes produce a standard mesh in the viewport that you can export or modify, its surface is made up of polygons which may appear too jagged at times, so switching to **Isosurface** mode would allow you to render very smooth surfaces which are calculated at render time.

The Simulator also takes care of rendering grid voxel data as a volumetric effect, which is the preferred mode for fire, smoke, clouds and explosions. Since liquid simulations can produce both particles and grid voxels, you could also render liquids as smoke volumes if you like.

Particle data is rendered by connecting a Particle Shader to the Simulator, and you can also use the Particle Shader to render standard particle systems that don't come from Phoenix. The Particle Shader is quite versatile and can draw particles as points, bubbles, or could also voxelize particles into a grid and render that as fog using the volumetric shader - this is how you can render mist from a waterfall, for example.

Just as you can import data from other simulators or content creation tools and render it using Phoenix's shaders, you can also use the Phoenix Simulator to convert the loaded data into certain formats which can be loaded by other software. This way you can do your rendering there, or you can use other simulation software to accompany Phoenix's simulations. Particles can be saved to the PRT format, which Phoenix can also import from other software using the PRT Reader node, and in turn you can plug that into the Particle Shader and render it. Phoenix can also export Alembic caches which contain particles and meshes, or VRMesh files containing only meshes.

Grid data can be loaded by Phoenix from its own AUR files, as well as from VDB and F3D files. Phoenix can also write particles during simulation to AUR and VDB files.

Rendering reads the data from the cache files and converts the physical data such as Smoke, Tmperature, Velocity, etc. into render data such as color and opacity. Note that the render settings can not be kept into the data cache files because they are different for each different renderer.

How to Set up a Simple Simulation

Phoenix comes with a toolbar that contains ready Quick Setup presets for commonly used fluid simulations.

The minimal setup needed for simplest simulation includes 3 objects: simulator, source helper and some geometry.

- Create a **simulator**.
- Create a geometry object and place it inside the simulator.
- Create a **source** helper.
- Select the geometry in the object list of the source helper.

At this stage, the setup is ready. To begin the simulation, go to the "Simulation" rollout of the simulator and press start.

Simulation

The simulation is performed in a box (called a grid) divided into small cells that contain the fluid's properties at their coordinates (temperature,velocity etc...). The minimal setup of the simulation core includes velocity and temperature as fluid properties of the fire/smoke simulation or velocity and liquid density for the liquid simulator, respectively. Depending on the core settings, one or more additional channels can be added and will be dragged along the fluid. These additional channels are:

- Speed
- Fuel
- Smoke
- RGB
- Wavelet
- Particle ID
- Particle Age
- Particle Size

The RGB channel does not affect the simulation; it's just a helper channel that can be used in the rendering for texture mapping or directly as RGB value. The Wavelet channel must be exported only when wavelet turbulence must be applied to the simulation on a second pass. Particle ID channel is helpful to identify the same particle between the frames.

The Simulation Workflow

The core recalculates the grid content repeating three main actions:

- Interact with external objects.
- Move the fluid with respect to the velocity (Advection).
- Change the velocity due to the internal collisions in the fluid (Conservation). Conservation is the same process which makes the liquid in a syringe's needle to move forward when the plunger is pressed.

Depending on the core settings, one or more additional operations can be included in the process. These operations are:

- Burning
- Vorticity Confinement

Burning Simulation

The burning process requires Fuel and Smoke as additional channels (they will be included automatically). Like in the real world, the burning process converts the fuel into smoke and produces some heat which expands the fluid. There is no separate channel for oxygen, however it is taken into consideration using the smoke and the fuel to be calculated. The relation is oxygen=1-fuel-smoke. The energy of the fuel controls how much heat will be produced and the inflation/expansion rate controls how fast the ignition will propagate.

For convenience the second parameter is just called "Propagation", but advanced users should keep in mind that the real meaning of the parameter is inflation/expansion rate.

Vorticity Confinement

The vorticity confinement keeps the small turbulences alive. It's natural for grid fire/smoke fluid solvers to lose fine details both in the density and in the velocity of simulations. Vorticity Confinement is a method for bringing these small details back and making the fluid more turbulent. However, classic vorticity tends to overcompensate and tear the fluid up too much, breaking the large-scale rolling of smoke and fire. In such cases, Phoenix's Massive Vorticity comes in handy - it does not allow the fluid to go wild and completely lose its shape and works well for both quick and slow moving fluids.

Interaction with External Objects

By default, the grid and particles in a simulation interact with scene objects. For example, in a liquid simulation, the liquid will bounce off or flow around any geometry it encounters.

There are three types of external objects that can interact directly with the simulator:

- Rigid bodies (polygon geometry)
- Particles
- Forces

In every simulation step, all the nodes in the scene are enumerated and processed whether or not they belong to one of the above mentioned groups and have permission to interact with the simulator.

Rigid Bodies

All geometry objects in the scene are considered to be rigid bodies. They can affect the simulation in three different ways:

- When the body is not selected as source but interacts with the simulator, the cells inside the geometry are frozen and the velocity of the surface cells is determined by the movement of the body.
- When the body is selected as source, the cells inside are frozen and the surface cells are set with the parameters of the source. The velocity is calculated according the discharge and the body's movement.
- When the body is selected as source, but doesn't interact with the simulator; the fluid inside the geometry is directly affected according to the **Emit Mode** of the simulator.

Another Phoenix simulator can also be used as Rigid body. In this case, the Surface Channel is used to determine the surface.

Particles

The particles can interact directly with the simulation only as sources, depending on the **Emit Mode** (**Volume Inject** or **Volume Brush**) of the Fire Source. The velocity of the injected fluid is the velocity of the particle, which allows the fluid to be involved in the particles' motion.

Forces

The forces change the velocity of the fluid in two different ways:

- If the force is a Gravity force, the acceleration is calculated directly using the buoyancy of the fluid. If the buoyancy is negative, the fluid accelerates toward the gravity force, otherwise it accelerates away from the gravity force.
- If the force is not Gravity, the fluid is accelerated toward the force. The magnitude of the acceleration is determined by the magnitude of the force.

Note: Max forces are slower than the included standard gravity and turbulence. Only some of the forces can work in multi-threaded mode.

Resimulation

Resimulation is the process of simulating over an existing simulation, which is called a base simulation. The base simulation must be processed first, and its Velocity channel must be exported. Then, to enable the resimulation, the **Enable** option in the Resimulation rollout must be checked. While this option is enabled, the simulator works in resimulation mode (i.e. all simulator functionality is now related to the resimulation).

The resimulation can affect the grid content and the particles separately. If only the grid is resimulated, the particles will be directly copied from the base simulation. If only the particles are resimulated, the grid content will be directly copied from the base simulation. Both grid content and the particles can be resimulated at the same time.

Note: Liquids do not currently support grid resimulation.

The resimulation process uses the velocity force of the base simulation to achieve the same flow of the fluid. This force can be resized if the user wants to increase the resolution when resimulating. There are two methods to resize the velocity force: **interpolation** and **wavelet turbulence**. The second method tries to add high frequency detail that is lost with simple **interpolation**, but it requires that the base simulation exports a special "Wavelet" channel and "Wavelet UVW" channel in addition to the regularly exported channels.

Since the resimulation can do sub-steps, it must do conservation of the velocity force on its own. However since this conservation is valid only inside a single frame, the global flow will not be affected, and lower conservation values can be used, which can greatly increase the overall simulation speed.

When resimulating, almost all simulator parameters can be changed (for example the SPF limits, advection step, time scale). However this can produce a flow of the fluid that is different from the original.

The **restore** function works with resimulation cache, even if the Velocity channel is not exported. However the base cache must be preserved.

Here are examples of what can be achieved with resimulation.

- Increase the resolution of an existing fire/smoke simulation, preserving its general flow.
- Add new channels/change source parameters of a fire/smoke simulation. Note that you may not get a physically accurate result. For example, with non-uniform conservation, the temperature affects the velocity.
- Increase/decrease the amount of drag particles, without doing a full simulation.
- Add foam and splashes, tweak parameters without fully simulating the liquid again.

Rendering

Rendering Mechanism

Phoenix registers itself as a global environment volumetric, except when working in Geometry Mode. The global volumetric takes care to blend properly all Phoenix instances in the scene (simulators and foam/splashes). However for overlapping with other volumetrics, geometry mode is recommended.

Channels, Diagrams and Gradients

After the simulation, cache files containing the exported physical channels are produced. In the real world, the connection between the physical quantities and the visual appearance is well-defined. However, for more flexibility in Phoenix, the user is allowed to choose how the render elements depend on the physics. For example, in the real world, the temperature determines the emissive color; however, in Phoenix, the velocity can be set to determine the emissive color instead of the temperature. The physical channel used to determine any given render element is called a **source**, and the dependence between the physical quantity and the render element is determined by tables called **palettes**. For the rendering of the volumetric content, three render elements are calculated for each point: emissive color, diffuse color, and opacity (alpha). To calculate each of them, the shader performs the following steps:

- Determines which physical channel is used as the source.
- Samples the input data in the shaded point to determine the value of the source channel.
- Passes the value of the source channel through the palette to obtain the value of the render element.

The control over the above mentioned process is arranged in Render Fire, Render Smoke Color, and Render Smoke Opacity rollouts where the user can choose the source and the palette for each channel. For the emissive color, both a diagram and a gradient are used, because–as opposed to the diffuse color–it can be outside the range 0-1. When this happens, the gradient will appear saturated. For each render element, the user is also allowed to select a texture map as the source channel. In this case, the palette is skipped and the sampled color is used directly.

Smoke Opacity vs. Fully Visible Shading

There is a well-defined connection between the transparency and the emissive light in the real world. For smoke (diffuse light), the connection is pretty clear and everyone has intuitive sense of how it works. For the emissive light, however, most of the people will give the wrong answer when asked, "Is a smokeless fire fully transparent?" The common, yet wrong, answer is, "Yes it is. If there is no smoke, the flame is fully transparent and will cast no shadows." However, the correct answer is "No, the fully transparent gases *cannot* emit any light, even if they are hot enough." This is why acetylene flame is less bright than a candle, despite its higher temperature. It just produces less smoke. However, this way of shading can seem strange and inconvenient, because there is no clearly separated control over the smoke and flame. Be careful when trying to achieve denser smoke, because the result can be an undesired brighter flame. For this reason, Phoenix provides different modes of shading: physical and detached. This is controlled by the **Fire Opacity Mode** parameter in the Fire rollout. In Fully Visible mode, the emissive part is fully independent of the transparency and you are not obligated to keep in mind the aforementioned connection. In Use Smoke Opacity mode, the smoke's alpha controls the Fire's opacity. There is also the Use Own Opacity mode where the fire has a separate opacity curve that can be adjusted independently from that of the smoke.

The Surface Channel

Some rendering techniques (like solid rendering, displacement, etc.) require a separate channel to work with, and for this reason the Surface channel is provided. Most of them use the Surface channel to calculate an implicit surface from its content, using the following rule: all the points with a value above given **threshold** will lie inside the surface and all the points with a value below the threshold will lie outside the surface. The Mesh Mode gives the best look over the whole concept, because it makes the surface visible.

Geometry Mode

Volumetric Geometry Mode

In this mode, the appearance of the content is exactly like in the Volumetric Mode, except for the ability to blend with other overlapping volumetric objects. However, it is slower to render and takes more memory. Also it requires the **Max. transp levels** in the Global Switches rollout in the V-Ray Render Settings to be increased.

Volumetric Heat Haze Mode

This mode is used to achieve mirage effects and heat haze. In this mode, the ray traced in the volume changes its direction at each step according to the content of the Surface Channel. It may appear that this mode is similar to pure geometry mode, but it is not. The shading is more similar to a V-Ray refractive material than the shading in volumetric mode or pure geometry mode. As with the shading of refractive materials, the alpha is set to 1 for the rays intersecting the simulator's grid.

Mesh Mode

This mode produces a solid object with a sharp surface, determined by the Surface channel, and can be shaded using any material.

Adding Fine Details

There are two direct methods that can be used to add fine details in the shading. In order to enhance the realism of the movement of small details in the fluid, V-Ray comes with a Particle Texture called Particle Texture. When used in combination with a particle system driven by a Phoenix Simulator, the particle texture is capable of properly animating the small details along the fluid surface.

Texture Modulation

This is the common way used in the fluid systems to add fine details. In the Render Fire, Render Smoke Color and Render Smoke Opacity rollouts is a parameter called **Modulate** near the texture slots for each render element. When enabled, the corresponding render element will be multiplied by the value of the texture map, except in the case when a texture is selected as the source channel.

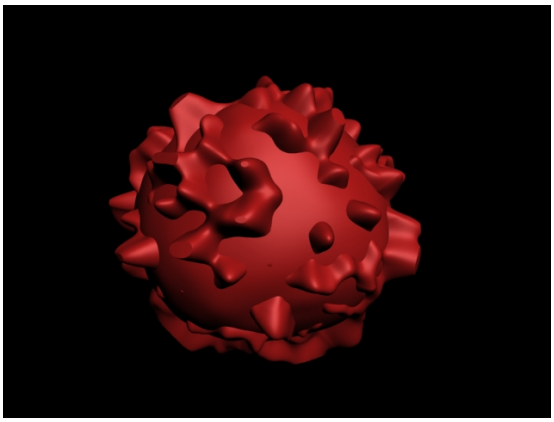
Displacement

This technique is more sophisticated than texture modulation and produces a significantly better result. The idea of displacement is similar to the usual geometry displacement, where the surface is displaced along its own normals at a distance determined by a texture map. The nearest point of the surface as specified by the Surface channel determines the direction for the displacement.

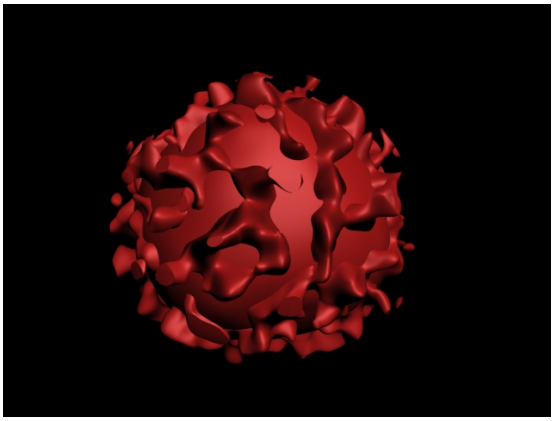
Surface Driven vs. Volumetric Displacement

The displacement in Phoenix works by shifting the point of sampling with an offset, the direction of which is determined by the gradient of the effects channel while the distance is determined by the brightness of a given map. There are two different modes used to sample the aforementioned map, switched by a checkbox in the rendering rollout. If the checkbox is NOT checked, the map is sampled at the point of shading, and we call this "volumetric displacement". If the checkbox is checked, the coordinates are first projected over the surface as determined by the effects channel. The point of projection is then used as map coordinates. We call this "surface driven displacement." Surface driven displacement is slower and requires some initial calculations, but it keeps the topology of the iso-surfaces. Volumetric displacement is faster, but tends to produce island-like formations if the displacement value is too big. Beside performance, volumetric displacement has one more important advantage: it does not require the distinct surface of the effects channel. Therefore, it will work fine in scenes where surface-driven displacement produces flickering.

Example: Surface Driven vs. Volumetric Displacement



Surface driven displacement



Volumetric displacement
Note the small islands floating above the original surface.

Foam and Splashes

The Phoenix solution for creating foam and splashes is particle based and is achieved by two relatively separate features - a foam/splashes particle simulator and a foam/splashes particle shader. The particle simulator is embedded in the Phoenix Simulator node and exports its content via **PhoenixFDP Group** nodes, supporting position, velocity, size and id channels. The particle shader is designed as a separate component: Particle Shader. This component can render **PhoenixFDPGroup** and any standard particle system that exports position, size and velocity channels.

Particles of the Phoenix Simulator

There are several particle types that the **Phoenix Simulator** exports:

- Foam
- Splashes
- Mist
- Drag

The birth of particles can be performed automatically (for foam and splashes) by a user source (see Fire Source) or by a script. When automatic birth is used, the simulator calculates a "birth potential" for each cell and compares this potential with the birth threshold set by the user. If the potential is above the threshold, the simulator creates new particles in the quantity specified in the "birth rate" parameter, in thousands per cubic scene unit. Both foam and splashes are born in this way; the only difference is how the "birth potential" is calculated. Additionally, when a splash particle hits the water surface, foam is automatically created.

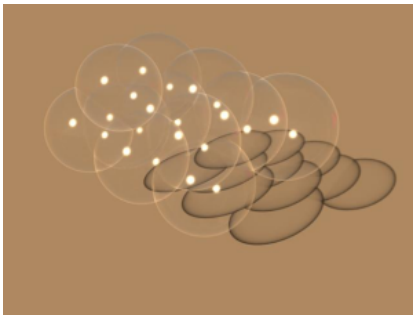
The simulation of the particles is very different for each type. The simplest one is the "drag". This type of particle simulation generates particles that are just dragged with the fluid. This is much like the **PhoenixFDForce** operator for the standard particle systems but much more efficient: you can drag a million particles in few minutes. The second type (by complexity) is the splashes type. The splashes are just left in free fall until they hit some object or the water surface. The free fall is not the simplest one, it is affected by the movement of the air (do not forget that Phoenix simulates the air too, not just the liquid!) and the air-splashes interaction is controlled by the air friction parameter. The most complicated type is foam. The foam particles interact not only with the fluid, but with each other as well. This interaction produces a repulsive force when the particles are too close, and an attractive force when they are not, ensuring the foam's consolidation. This process is the most expensive one and is controlled by the **Foam Volume** parameter. When only surface foam is needed, the aforementioned process can be switched off by setting the parameter to zero.

Rendering of Foam and Splashes

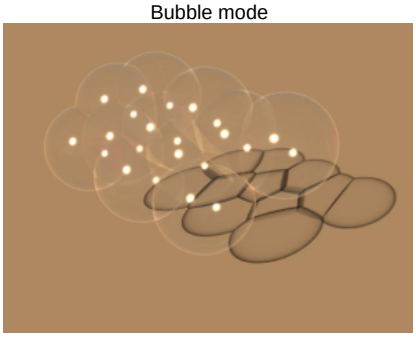
The Particle Shader node can render the particle systems in several different ways - as separate bubbles/droplets, as points, or as fog. In fog mode, the bounding box of all particles is calculated and a grid based shader is constructed using the bounding box and the fog resolution parameter. This mode is most suitable for surface foams, when the grid height is relatively large and the particle count is huge. The other mode (particle-based shading) is more complicated and interesting; it allows us to achieve a larger variety of effects. One reasonable question about this mode is why we need it, when one can render each particle as geometry with the proper material instead. There are several advantages, but the most important one is the ability to convert the scattering into diffuse color, which gives the natural bright appearance of the foam and the splashes. Another important advantage is cellular mode, which allows shading of close-up foam.

There are three different modes of shading for spherical particles - bubbles, cellular, and splashes. In each mode, a particle is represented as a sphere with a radius equal to its size and with certain optical properties. The "bubble" and "cellular" modes are intended for foam rendering and have the same optical properties of the surface. They differ in the geometry representation only - bubble mode uses simple spheres, whereas cellular mode builds polyhedron-like cells, similar to real foam. The cell walls are not flat, but slightly curved, which provides a more realistic appearance.

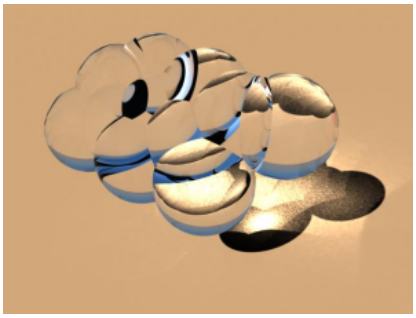
Example: Rendering of Foam and Splashes



Bubble mode



Cellular mode



Splashes mode