

FireSmoke Dynamics

This page provides information on the Dynamics rollout for a FireSmokeSim object.

Overview

This rollout controls the FireSmoke dynamics parameters, which affect the fluid's behavior when simulating. The Dynamics rollout can be accessed in the Modify panel when a FireSmokeSim object is selected.

UI Path: ||Select Fire Smoke Simulator|| > **Modify panel > Dynamics rollout**

Parameters

General

Expand – Opens a floating dialog that contains the selected rollout and automatically folds the command panel rollout.

Re-Center – Resets the position of the floating rollout.

? – Opens up the help documents for the Fire/Smoke Dynamics.

Motion Inertia | *ext_wind, inert_mul* – When enabled, moving the simulator object over a series of frames causes inertial forces in the opposite direction of the movement. This allows you to link the simulator to a moving object and keep the size of the grid relatively small, as opposed to creating a large grid that covers the entire path of the moving object. **Motion Inertia** can be used for moving ground and water vehicles, torches, fireballs, rockets, etc.

Gravity | *grav, gmul, extgrav* – Phoenix **Gravity** makes the liquids fall down and makes fire rise up. In fire/smoke simulations it creates velocities depending on the Temperature channel. In voxels where the Temperature is above 300 (in Kelvins, which is room temperature), **Gravity** will create upwards velocities. Where Temperature is below 300, the opposite will happen - velocities will be created pointing downwards. The hotter the fluid, the faster it will rise, and the cooler the fluid below 300, the faster it will fall down. The **Gravity** option is a multiplier, so using the default 1.0 will make it behave like real world gravity, setting it to 0.0 will disable its effect completely, and you can also use negative values, which will inverse the gravity effect.

Time Scale | *timescale* – Specifies a time multiplier that can be used for slow motion effects. In order to achieve the same simulation look when changing the time scale, the **Steps Per Frame** value must be changed accordingly. For example, when decreasing the **Time Scale** from 1.0 to 0.5, **Steps Per Frame** must be decreased from 4 to 2. And of course, all animated objects in the scene (moving objects and sources) must be adjusted as well. **Time Scale** different than 1 will affect the **Buildup Time** of Particle/Voxel Tuners and the Phoenix Mapper. In order to get predictable results you will have to adjust the buildup time using this formula:

Time Scale * Time in frames / Frames per second

Cooling | *cooling* – This parameter controls the cooling of the fluid due to radiation. It gradually decreases the temperature until it reaches 300.

Phoenix temperatures are in Kelvin, so 300 is room temperature - the temperature where the smoke neither ascends nor descends. A Cooling value of 1.0 corresponds approximately to the speed real smoke with a half thickness of 4 meters cools down. The real cooling is a very complicated process similar to the Global Illumination rendering, so here a simplified formula is used. You can find out more about Phoenix Grid Channel Ranges [here](#).

Smoke Dissipation | *smdiss* – Used in simulations where the smoke needs to disappear, for example steam, clouds, etc. If you set this parameter to the maximum value of 1, the smoke will disappear immediately.

Smoke Buoyancy | *smoke_bcy* – The buoyancy of the smoke. Positive values make the smoke move upwards. Negative values make it move downwards.

Fuel Buoyancy | *fuel_bcy* – Specifies the buoyancy of the fuel.

Expand
Re-Center
?

<input checked="" type="checkbox"/> Motion Inertia	1.0	⬆️⬆️
<input checked="" type="checkbox"/> Gravity	1.0	⬆️⬆️
Time Scale	1.0	⬆️⬆️
Cooling	0.17	⬆️⬆️
Smoke Dissipation	0.0	⬆️⬆️
Smoke Buoyancy	0.0	⬆️⬆️
Fuel Buoyancy	0.0	⬆️⬆️

Vorticity

Classic Vorticity | *vorticity* – Adds small-scale detail that is dissipated naturally by the grid-based simulation. Prevents the simulation from becoming smooth and laminar. Unlike [Turbulence](#), which does not care about the fluid's motion at all, the **Classic Vorticity** algorithm works depending on the velocity of the simulation and changes the velocity field in order to reinforce vortices and add more detail to the simulation. *For more information, see the [Vorticity example](#) below.*

Massive Vorticity | *velvort* – This mode generally improves the vorticity detail and behavior. Unlike **Classic Vorticity** which reinforces vortices throughout the grid, the **Massive Vorticity** gets applied more strongly only to some parts of the fluid, making a difference between thin air with no smoke or temperature, the smoke and temperature's surface, and the inside volume of fluids. The added detail is equally strong for both the fast and the slow moving parts of the fluid. **Massive Vorticity** also does not produce a boiling effect that is present in the **Classic Vorticity** algorithm, which tears up the slower moving parts of the fluid excessively and stops the fluid's motion. *For more information, see the [Vorticity example](#) below.*

Smoke Surface/Temp. Surface | *vorticitySurfSm, vorticitySurfT* – These options create vortices only on the smoke/temperature channel's surface without disturbing the flow in the inner volume of the fluid. This way massive vorticity won't block the fluid's general motion and break the fluid apart as the classic vorticity does. It's split into separate smoke and temperature multipliers for additional flexibility. For example, in a fire-only simulation, you need to use **Temp. Surface**, while in a smoke-only simulation, you need **Smoke Surface** instead.

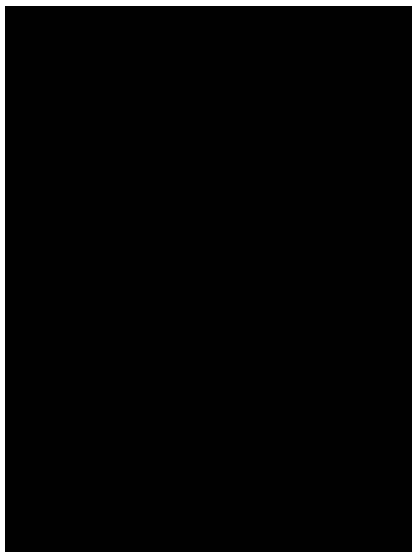
Large Scale | *vorticity_low* – The large-scale multiplier reinforces the big vortex flows in the inner part of the fluid. The effect is similar to what higher **Fluidity** quality can achieve, with the difference that the **Large Scale** vorticity does not respect obstacles, so it might not be suitable in a more complicated setup. *For more information, see the [Vorticity example](#) below.*

Vorticity

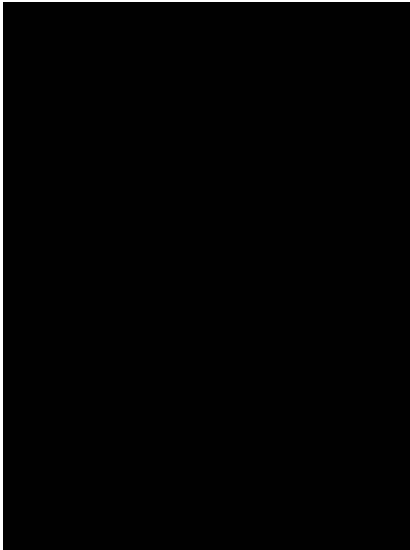
Classic Vorticity	0.3	▲▼
<input checked="" type="checkbox"/> Massive Vorticity		
Smoke Surface	0.3	▲▼
Temp. Surface	0.1	▲▼
Large Scale	0.5	▲▼

Example: Vorticity

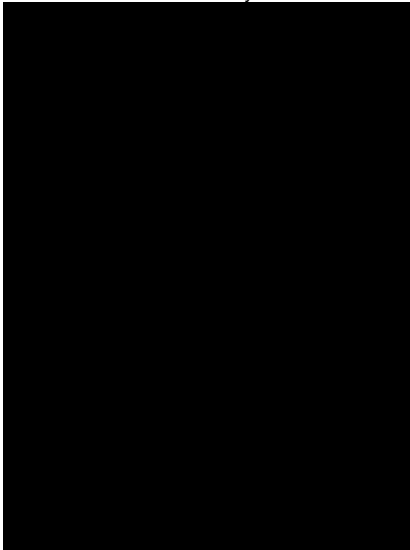
The following example shows the difference between the Classic and Massive types of Vorticity.



Classic Vorticity = 0



Classic Vorticity = 0.3



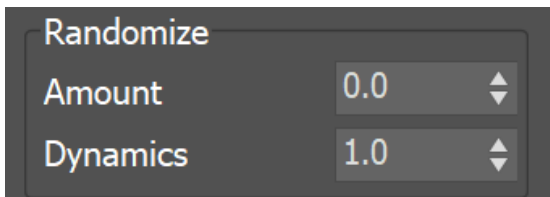
Massive Smoke Vorticity = 0.3
Large Scale = 0.3

Randomize

These options add random fluctuations in the fluid's velocity for each grid voxel.

Amount | *rand* – The amount of randomization. Only the velocity length is changed; the velocity direction remains the same.

Dynamics | *mdfrq* – The speed of change of the fluctuations. The higher the value, the faster the random noise will change over time.



Fluidity (Conservation)

The conservation process gives the fluid its characteristic swirling motion. It transforms straight-line movement of the fluid into swirling vortices. The higher the strength of the conservation, the farther the motion forces will be propagated throughout the container, so a movement in one point will cause the fluid to start moving at a distance too. The conservation directs smoke and fire into realistic shapes and helps liquids to support their own weight when at rest, and to fill up a volume they are poured into.

Internally, the conservation updates the directions and magnitudes of the velocities of each cell in the grid, preparing them for the Advection step when the content will be moved between cells. Basically it tries to equalize the velocities coming in and going out from each cell, and does this in many passes, getting closer to the perfect equilibrium. The number of passes is the conservation strength (quality). In nature, conservation has an infinite strength and is always perfect. In Phoenix, the better the quality of the conservation is, the farther the movement from one point will be propagated, making the simulation more realistic, but at the expense of longer simulation time for each frame. There are a number of conservation methods in Phoenix that you can choose between, depending on the type of your simulation. Each of them comes with pros and cons for the given situation.

Method | *refltype* – Specifies the method for simulating the conservation. *For more information, see the [Conservation Method Types](#) example below.*

Direct Symmetric – This method has long range and can quickly transfer velocity and pressure from one end of the grid to the other end. It also achieves strong conservation without having to boost the quality too much, and thus the fluid can roll and swirl quite well. Thus it is well suited for simulation of explosions with shockwaves.

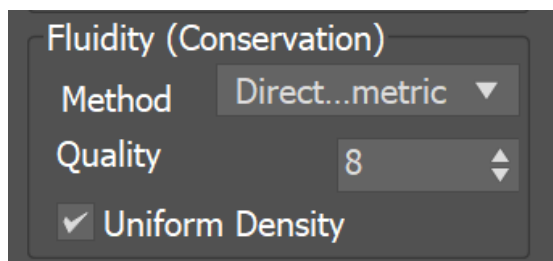
Direct Smooth – The smooth method is quite similar to **Direct Symmetric**. It produces a smoother velocity field and is stronger than **Direct Symmetric**, but doesn't keep the symmetry that well and it will produce less detail.

Buffered – This method has the weakest conservation strength and shortest range, but produces detail that works well when simulating fire.

PCG Symmetric – This method provides a very strong conservation, but works at a shorter range than the **Direct Symmetric** method. It actually produces better symmetry than **Direct Symmetric** and so it can be used to simulate nuclear mushrooms or other effects where high symmetry is crucial. It is also the best method to use for smoke or explosions in general. The **Uniform Density** option is ignored in this mode, so simulating only fire with this mode may not produce as good results as using the other methods that allow **Uniform Density** to be enabled.

Quality | *reflprec* – Increases the strength of the conservation. It will help cases when the liquid or smoke is losing volume, or the swirling of the smoke needs to be increased. Beware that increasing the quality will slow down the simulation. *For more information, see the [Quality example](#) below.*

Uniform Density | *uniform_mass* – When enabled, the mass of the fluid will not be considered. Fire simulations work better with this option on, which ignores the mass. However, unchecking this option might be useful for pure smoke simulations or explosions (for fire/smoke simulations the inverse of the temperature is considered the mass; the hotter fluid will be lighter and the cooler fluid will be heavier, just as in nature).



Example: Conservation Method Types

The following example shows the difference between the Symmetric, Smooth and Buffered Conservation Method types.



Symmetric conservation



Smooth conservation



Buffered conservation

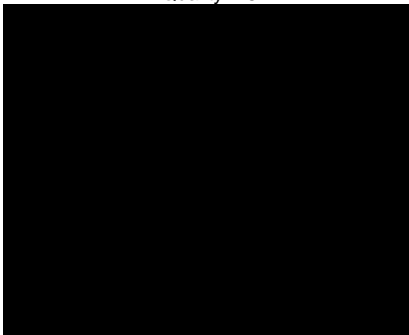
Example: Conservation Quality

The animation below demonstrates how to cope with losing the volume of smoke. A smoke source is placed in an almost closed room, where the only exit is near the floor. In the real world the smoke will fill the room and then reach the exit, just because once created, the smoke is accumulated and does not disappear. However, when the conservation parameter is low, the smoke does not keep its amount and disappears, never reaching the hole.

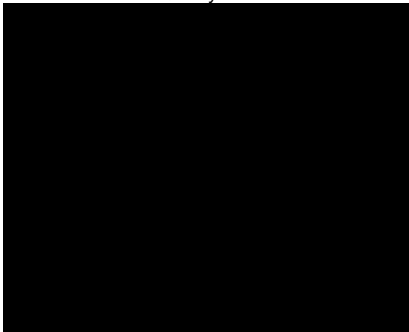
Note that the smoke reaches the hole with conservation quality at about 200, and this value is not accidental. The vertical size of the grid is 100, and if it was smaller, the conservation value that allows the smoke to reach the hole would have been smaller too. You can think of this parameter as a distance scope in which the conservation spreads the velocity influence.



Quality = 8



Quality = 50



Quality = 200

Transport (Advection)

Advection is the process that moves the fluid along its velocity inside the grid. A problem of all grid-based simulators is that moving the content of one cell to a new place in the grid will blur the result when the destination lies between cells (and it usually does), thus losing the fine details with each new frame. Phoenix has a number of advection methods that battle this problem in different ways, each with its own pros and cons depending on the situation.

Phoenix may perform advection more than once per frame, or once in a number of frames, depending on the **Steps Per Frame (SPF)** parameter.

When you press simulate, the activity of the fluid in all of the voxels is calculated in sequential steps, representing the passage of time. The order of steps is always sequential, meaning the simulator will calculate fluid properties sequentially from one step to the next, until you have a series of steps that map out the fluid's behavior over time.

The number of steps can be modified using the **Steps Per Frame (SPF)** parameter, which determines how time is subdivided, and has a significant impact on the behavior, quality and performance of the simulation.

To get the most detail for fire and smoke simulations, it is best to keep the **Steps Per Frame (SPF)** low. On the other hand, a higher **SPF** works better to keep liquid simulations smooth and steady, and can also produce better results for fast moving fluids in general.

Method | *advtype* – Specifies the algorithm used for calculating the advection. *For more information, see the [Advection Method Types](#) example below.*

Forward Transfer – This method can help if you are losing fluid volume in more rapid simulations, but is less detailed than Multi-Pass and could also be less stable.

Multi-Pass – This less dissipative method produces more fine details and keeps the smoke interface sharper compared to other methods. This method is recommended for large scale explosions, veil-like smoke, pyroclastic flows, and all other situations where sharpness is important.

Steps Per Frame (SPF) | *spf* – Determines how many calculations the simulation will perform between two consecutive frames of the timeline. *For more information, see the [Steps Per Frame](#) example below.*

Steps Per Frame (SPF) is one of the most important parameters of the simulator, with a significant impact on quality and performance. To understand how to use it, keep in mind that the simulation is a sequential process and happens step by step. You cannot take a shortcut to simulate the last frame of a simulation, without first simulating all of the frames that come before it, one by one.

The simulation produces good results if each step introduces small changes to the sim.

For example, if you have an object that is hitting a liquid surface with a high speed, the result will not be very good if at the first step, the object is far away from the water, and at the second step, the object is already deep under the water. You need to introduce intermediate steps, until the object's movement becomes small enough that it happens smoothly across all steps for that frame.

The **SPF** parameter creates these steps within each frame. A value of 1 means that there are no intermediate steps, and each step is exported into the cache file. A value of 2 means that there is one intermediate step, i.e. each second step is exported to the cache file, while intermediate steps are simply calculated, but not exported.

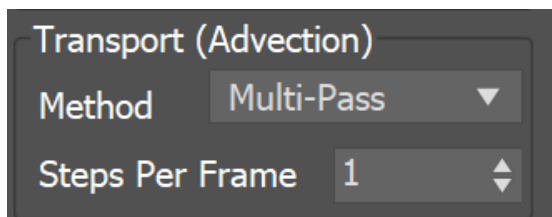
Increasing the **Steps Per Frame (SPF)** also comes with significant trade-offs to performance and detail.

A higher SPF decreases performance in a linear way. For example, if you increase the **SPF** twice, your simulation will take twice as long. However, quality does not have a linear relation to **SPF**.

For maximum detail, it is best to use the lowest possible SPF that simulates without any of the issues described in the tip box below, since each additional step kills fine details. For more information, please refer to the Phoenix Explained docs. Signs that the **Steps Per Frame (SPF)** needs to be increased include:

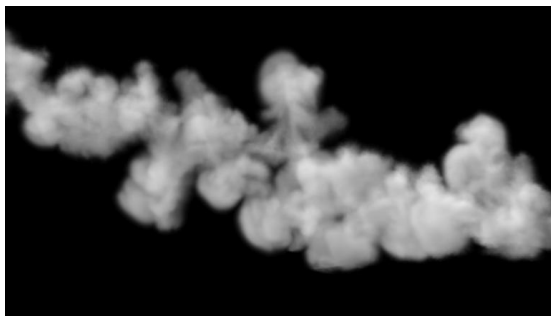
- Liquid simulations that have too many single liquid particles.
- Liquid simulations that appear torn and chaotic.
- Liquid simulations of streams that have visible steps or other periodical artifacts.
- Fire/Smoke simulations with artifacts that produce a grainy appearance.

More often than not, these issues will be caused by the simulation moving too quickly (e.g. the emission from the source is very strong, or the objects in the scene are moving very fast). In such cases, you should use a higher **SPF**.

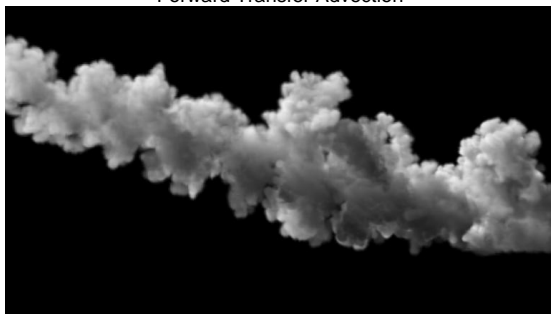


Example: Advection Method Types

The following example shows the difference between the Classic and Multi-Pass advection types.



Forward Transfer Advection



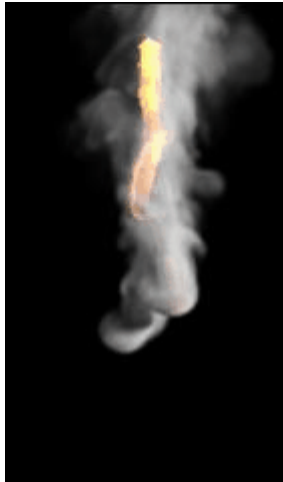
Multi-Pass Advection

Example: Steps Per Frame (SPF)

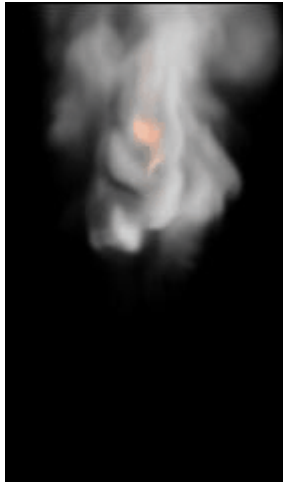
The first series in this example shows the differences in a Fire/Smoke simulation when the **Steps Per Frame** is set to 1, 2, and 8.



SPF = 1



SPF = 2



SPF = 8

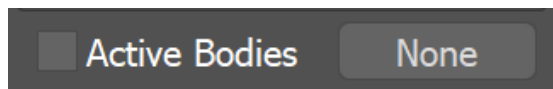
Active Bodies

Interaction between Active Bodies and the Phoenix Fire/Smoke Simulator is not supported yet.

For more information on Active Bodies, please check out the [Active Body Solver](#) and the [Active Bodies Setup Guide](#).

Active Bodies | *use_activeBodySolverNode* – Enables the simulation of Active Bodies.

Active Body Solver | *activeBodySolverNode* – Specifies the [Active Body Solver](#) node holding the objects to be affected by the Phoenix Simulation.



Texture UVW

The main purpose of the Texture UVW feature is to provide dynamic UVW coordinates for texture mapping that follow the simulation. If such simulated texture coordinates are not present for mapping, textures assigned to your simulation will appear static, with the simulated content moving through the image. This undesired behavior is often referred to as 'texture swimming'.

UVW coordinates are generated by simulating an additional **Texture UVW Grid Channel** which has to be enabled under the **Output** rollout for the **settings below to have any effect**.

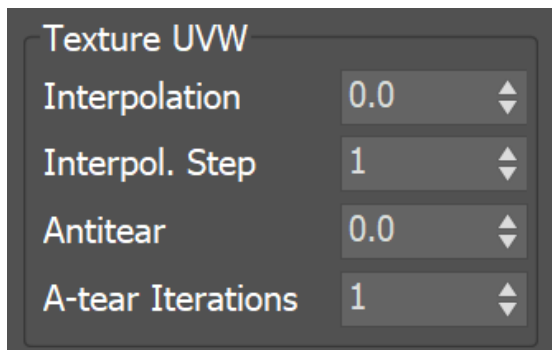
The custom UVW texture coordinates can be used for advanced render-time effects, such as recoloring of mixing fluids, modifying the opacity or fire intensity with a naturally moving texture, or natural movement of displacement over fire/smoke and liquid surfaces. For more information, please check the [Texture mapping, moving textures with fire/smoke/liquid, and TexUVW](#) page.

Interpolation | *texuvw_interpol_influence* – Blends between the UVW coordinates of the liquid particle at time of birth and its UVW coordinates at the current position in the Simulator. When set to 0, no interpolation will be performed - as a consequence, textures assigned to the fluid mesh will be stretched as the simulation progresses. This is best used for simulations of melting objects. When set to 1, the UVW coordinates of the fluid mesh will be updated with a frequency based on the **Interpol. Step** parameter - this will essentially re-project the UVWs to avoid stretching but cause the textures assigned to the fluid to 'pop' as the re-projection is applied. If you intend to apply e.g. a displacement map to a flowing river, set this parameter to a value between 0.1 and 0.3 - this will suppress both the effects of stretching and popping. See the [Interpolation example](#) below.

Interpol. Step | *texuvw_interpol_step* – Specifies the update frequency for the UVW coordinates. When set to 1, the UVWs are updated on every frame, taking into account the **Interpolation** parameter. See the [Interpolation Step](#) example below.

Antitear | *texuvw_antitear_influence* – Use this option when the assigned texture appears twisted, torn apart or otherwise distorted. This may happen when the simulation is moving very fast, therefore increase both the **Antitear** and **A-tear Iterations** to let Phoenix attempt to resolve the distortion.

A-tear Iterations | *texuvw_antitear_iterations* – The number of **Antitear** iterations performed for every Step of the simulation. Increasing this parameter will help resolve UVW distortion issues by allowing Phoenix to run the **Antitear** operation multiple times. Note that this may slightly increase the time it takes for the simulation to complete.



Example: Interpolation

The following video provides examples to show the differences of **Interpolation** values of 0, 0.1, and 1, and an **Interpolation Step** of 1.

Example: Interpolation Step

The following video provides examples to show the differences of **Interpolation Step** values of 1, 3, and 6.