

SKYLIFE emFluid v. 1.0

by Eric Mootz, summer 2007.

1. Introduction

The XSI command *emFluid* is a plug-in to simulate the behaviour of fluids and gases. The command *emFluid* is called within a particle event, which means that most of XSI's particle functionality (forces, obstacles, ..) can be used together with the command.

Requirements: *emFluid* works with XSI Version 5.11 and above, Foundation, Essentials and Advanced.

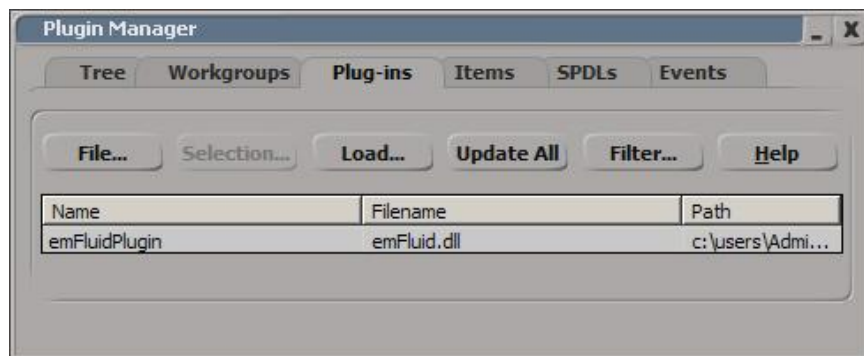
2. Installation

a) Copy the file "emFluid.dll" into XSI's user plug-in directory, e.g. "C:\users\Administrator\Softimage\XSI_x.xx\Application\Plugins". This file is the actual program that calculates the fluid behaviour.

b) Copy the file "emFluid_PEvent.vbs" into XSI's installation plug-in directory, e.g. "C:\Softimage\XSI_x.xx\Application\Plugins". This script is called within a scripted particle event of a particle cloud. It will read all the user's settings and execute the emFluid command.

Note: the file "emFluid_PEvent.vbs" actually can be located wherever you want (e.g. in a workgroup directory). If you do so, the scripted particle event might report an error, saying it can't find the file. This is no problem: you simply need to relocate the file (also see chapter 6. "Trouble shooting").

Once the files are copied you can start XSI and open the Plug-in Manager (File -> Plug-in Manager...), then click on the tab *Plug-ins*. You now should see something like this:



If the *emFluid* plug-in is listed, as in the picture above, then XSI now knows the command "*emFluid*" and we are ready to use it.

Hint: if, at some point, you have problems working with *emFluid*, please go take a look at the "trouble shooting" section. Actually, even when you are *not* having any problems you can look there (lots of useful information!).

Demo Version: the demo version of *emFluid* has the following restrictions:

- You cannot have more than two fluid boxes at a time.

- You cannot have more than two velocity forces per fluid box.
- The option “Velocity Force from Geometry (Point Normals)” has no effect.
- *emFluid* will only work from frame 0 to 300.

3. *emFluid* in action: some examples

Before we start looking at how exactly *emFluid* works and how to use it, let’s take a look at a few example scenes that use the *emFluid* plug-in. All the scene files are located in the XSI database “emFluid_db”.

Example 1: a simple fire

Load the Scene “inAction01_Fire.scn”. Play it back from the beginning. The scene will probably play back in less than 25 fps the first time, because XSI and *emFluid* have to calculate the particles. Once the playback is done, play back the scene again. It should now play back in 25 fps without any problems on a “normal” machine (e.g. P4 2.7GHz, 512 MB RAM).

Note: if you are using the demo version, then the *emFluid* - simulation will stop at frame 300.

Important: it is very important to simulate the particles by playing back the scene from the beginning. If you directly go to the last frame then *emFluid* might not produce the desired results! There is more information about this in the section “Trouble shooting”.

Example 2: a simple fire with two XSI Obstacles

Load the Scene “inAction02_FireWithObstacles.scn”. It is more or less identical with the scene of the first example. This time, though, there are two XSI obstacles included.

Play the scene back from the beginning and, when done, play it back again to have a decent frame rate of 25 fps.

Note: if you are using the demo version, then the *emFluid* - simulation will stop at frame 300.

Example 3: fire and steam

Load the Scene “inAction03_CookingXSI.scn” and play it back.

In this example, we have two particle types, an obstacle and a XSI force (drag).

Note: if you are using the demo version, then the *emFluid* - simulation will stop at frame 300.

Example 4: basic smoke explosion

Load the Scene “inAction04_SmokeExplosion.scn” and play it back.

Example 5: basic smoke explosion

Load the Scene “inAction05_DirectSmokeExplosion.scn” and play it back.

Note: if you are using the demo version, then only two of the five velocity forces are used, but this example will look okay anyway.

Example 6: clusters of balls that meet

Load the Scene “inAction06_TwoIntersectingBallclusters.scn” and play it back.

Example 7: three smoke explosions

Load the Scene “inAction07_ThreeSmokeExplosions.scn” and play it back. It will automatically loop. Here we have three smoke explosions as in example 5 and a moving sphere obstacle.
Tip: select the particle cloud to display the particles as points with velocity trails.
Note: if you are using the demo version, then only two of the five velocity forces are used.

Example 8: curious noise

Load the Scene “inAction08_CuriousNoise.scn” and play it back. Here we have no velocity force, only velocity randomness.

Example 9: velocity force from geometry (1)

Load the Scene “inAction09_VelForceFromGeom1.scn” and play it back. Here we have a polygon mesh that emits particles and that also is a “velocity force from geometry”.

Example 10: velocity force from geometry (2)

Load the Scene “inAction10_VelForceFromGeom2.scn” and play it back. Here we have a polygon mesh that emits particles and that also is a “velocity force from geometry”.

4. Tutorials

All the scene files and scripts are located in the database “emFluid_db”.

Hint: you can view this file in XSI’s NetView.

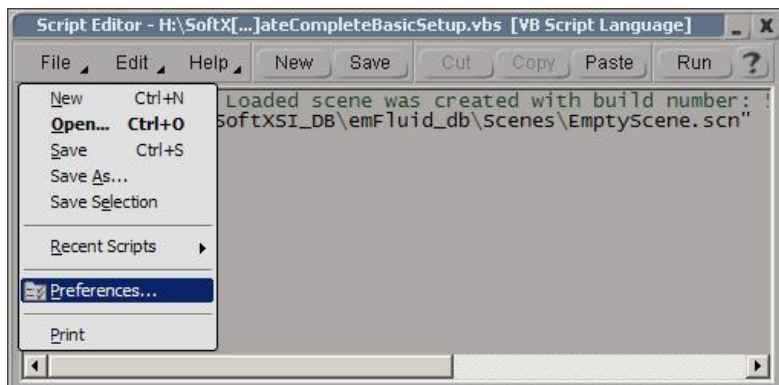
4.1. Tutorial 1: playing with a velocity force

a) Load the scene “EmptyScene.scn”. This scene contains nothing except for the default camera and light. Some of the camera’s display options are set to have a nice display in the camera view (view B) of the particles and objects involved.

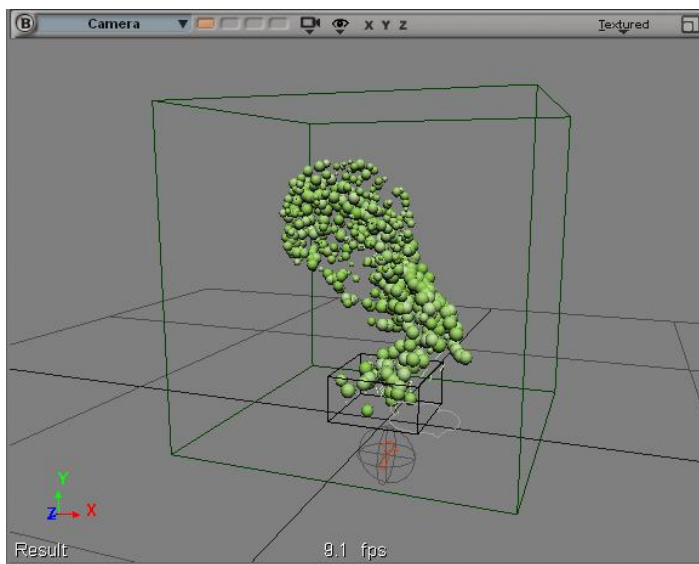
b) Open the script editor and load the Visual Basic script “emFluid_CreateCompleteBasicSetup.vbs” that is located in the folder “Scripts” of the database “emFluid_db”.

The script will create the complete basic setup for animating particles with the *emFluid* plug-in.

Important: Make sure the script language in the script editor is set to “VB Script Language”. You can check this in the preferences of the script editor (File->Preferences”):

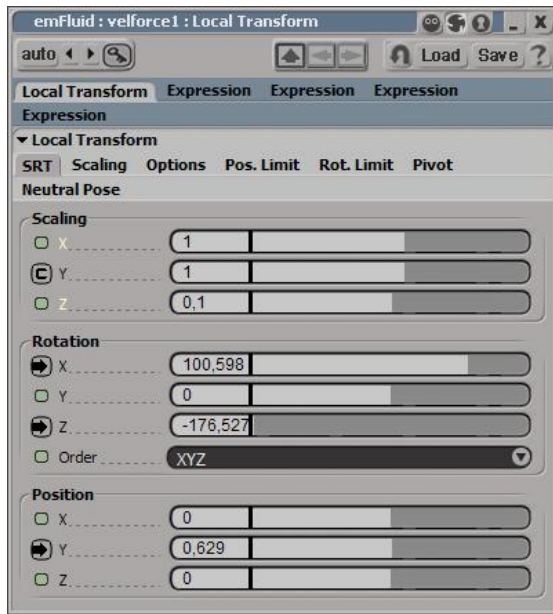


c) Run the script. Choose “Yes” when asked if you want a default animation. When the script is done, close the message box and the script editor. Play back the scene. You should see some balls move around in the big box:

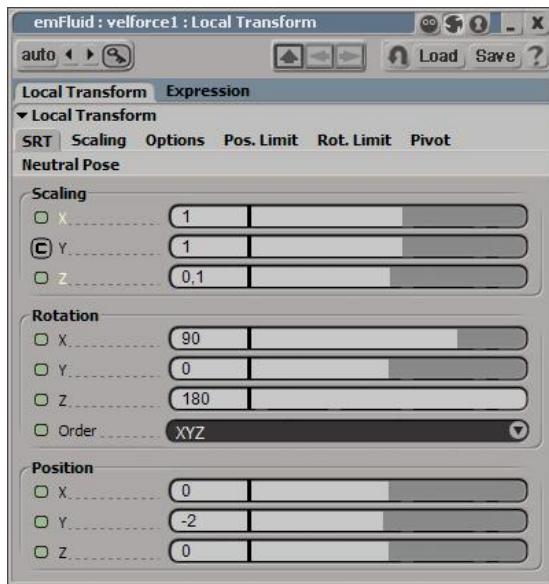


The big box is a “**fluid box**”, in which *emFluid* moves the particles. The orange arrow that bounces is a so called “**velocity force**”. It adds velocity to the fluid box. The velocity force is a simple null. Its position tells *emFluid* where it must add velocity, its local z axis defines the direction of the velocity and its x scaling defines the “brush size”.

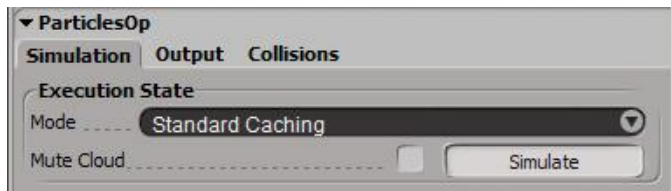
d) Let’s animate the velocity force manually: select the velocity force “velforce1” and open the property page of the local kinematics:



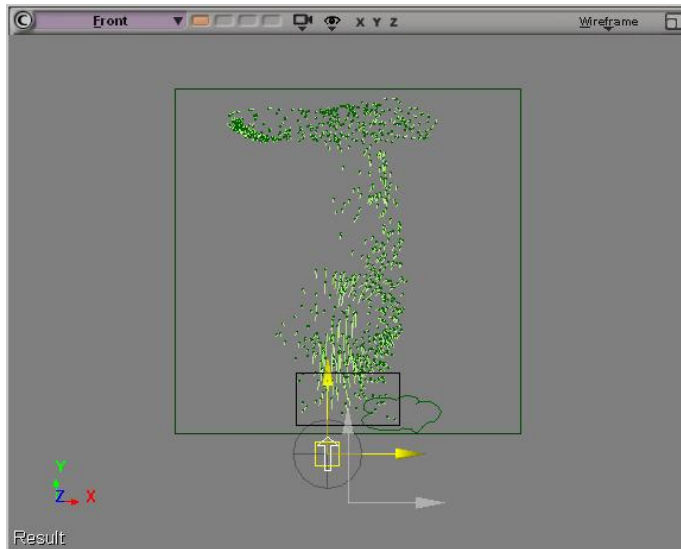
Remove all animations except for the one of the y scaling and set the x rotation equal “90”, the z rotation equal “180” and the y position equal “-2”:



e) Go to the first frame and reset the particle cloud by clicking on “Simulate” in the cloud’s particle operator:



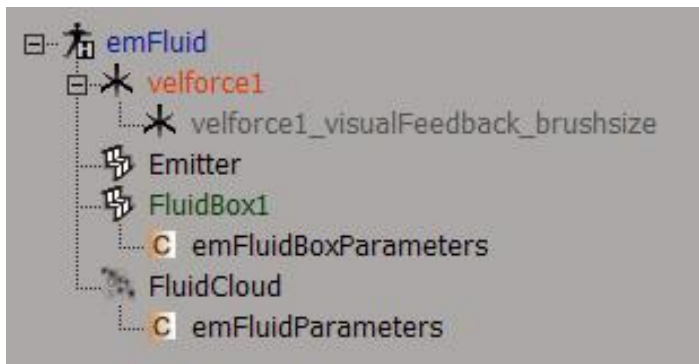
f) Now select the velocity force “velforce1”, activate the Translation Tool and start playback (play each frame). While the simulation is running, move the velocity force (front view is best) into the fluid box and out. The velocity force adds velocity at each frame, so the longer it stays in the fluid box, the more velocity is added:



Experiment by moving the velocity force more or less fast in and out the fluid box or by adding velocity not in the middle of the fluid box but at the sides. Also, you can change the velocity force's x scaling (brush size) and the z scaling (velocity strength), or rotate the velocity force to change the velocity's direction.

4.2. Tutorial 2: *emFluid* “parameter sight-seeing”

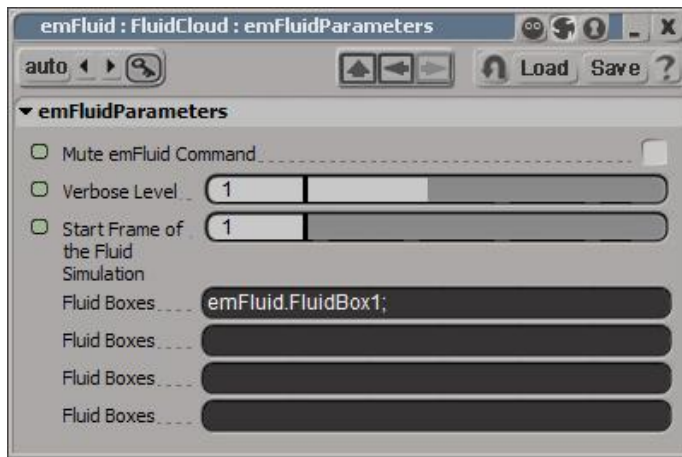
- a) Again, load the scene “EmptyScene.scn”.
- b) Open the script editor and load the Visual Basic script “emFluid_CreateCompleteBasicSetup.vbs”.
- c) Run the script. Choose “Yes” when asked if you want a default animation. When the script is done, close the message box and the script editor.
- d) Let's take a rough look at what the script generated:



- 1) A model called “emFluid” under which the rest is parented.
- 2) A particle cloud called “FluidCloud” with a custom parameter set called “emFluidParameters”. Any particle in this cloud can be influenced by *emFluid*.
- 3) A particle emitter called “Emitter”, which constantly emits particles at a rate of 250 particles per second.

- 4) A polymesh cube called “FluidBox1” with a custom parameter set called “emFluidBoxParameters”. The cube (or fluid box) defines the volume in which particles will be influenced by *emFluid*.
- 5) A null called “velforce1” with a child null called “velforce1_visualFeedback_brushsize”. The null “velforce1” is a sort of brush with which you can add velocity to the fluid box. The x scaling defines the brush size, the z scaling defines the amount of velocity and the local z axis defines the direction of the velocity. The purpose of the null “velforce1_visualFeedback_brushsize” only is to give some visual feedback concerning the brush size (= x scaling).
- 6) The script also created a particle type with a scripted particle event. The script of the particle event gathers the values of the custom parameter sets and calls the command *emFluid*.

e) Open the property page of the custom parameter set “emFluidParameters” of the cloud “FluidCloud”:



The parameter set “emFluidParameters” has following parameters:

Mute emFluid command: if checked, then *emFluid* is muted and won’t have any influence on the particles.

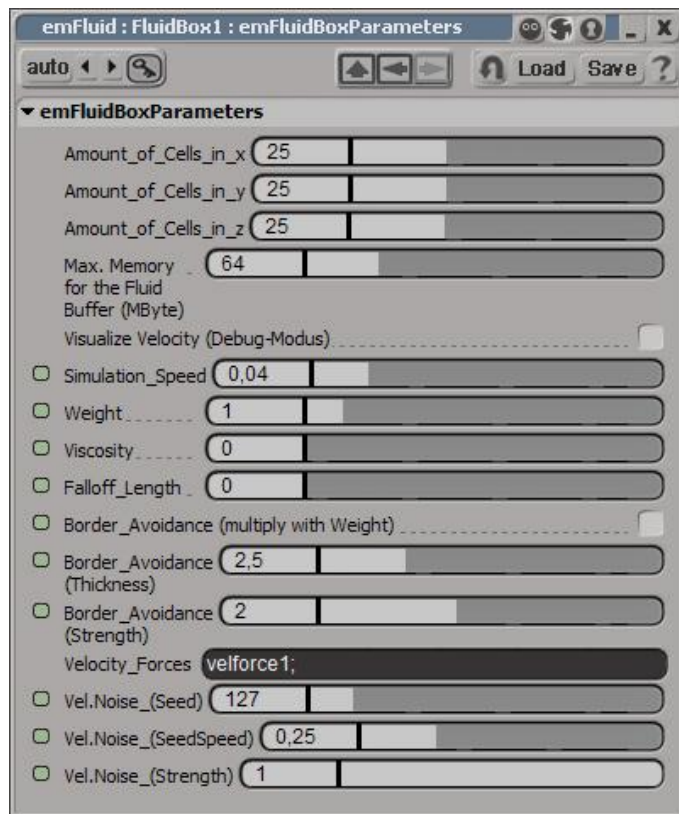
Verbose Level: this sets the verbose level of *emFluid*. All output of *emFluid* can be found in the history log of the Script Editor.

Start Frame of the Fluid Simulation: this is the frame at which the fluid buffers are reset.

Fluid Boxes: here you enter the names of the objects (separated by ‘;’) that shall be used as fluid boxes.

Note: there is more detailed information about this parameter set and its parameters in section 5.1.1.

f) Open the property page of the custom parameter set “emFluidBoxParameters” of the object “FluidBox1”:



You have the following parameters for this fluid box:

Amount of cells in x/y/z: The volume (or space) given by the fluid box is divided into voxels. With these three parameters you define the subdivision in x, y and z of the space and therefore the number of voxels. Greater values require more memory and longer calculations.

Max. Memory for the Fluid Buffer (Mbyte): Defines the upper limit of memory, in Megabytes, the fluid buffers may require.

Visualize Velocity (Debug-Modus): When on, the particles are positioned in the centres of the voxels and have the velocity of the voxel.

Simulation Speed: This defines the speed of the fluid simulation. To produce correct results set this value to 1 divided by the frame rate of your scene.

Weight: This defines how strong the fluid box will influence the particles that are contained in it. A value of 1 means full influence, a value of 0 means no influence at all.

Viscosity: This defines the viscosity (the “liquidness”). Higher values give a more syrup – like effect.

Falloff Length: Defines the thickness, in Softimage Units, of a border surrounding the fluid box in which the influence of the fluid box will gradually become smaller and smaller.

Border Avoidance: Border avoidance means: the nearer a particle gets to the outer limit of the fluid box, the more it will want to get away from the border.

Multiply with Weight: if checked, then the amount of border avoidance is multiplied with the Weight parameter.

Thickness: the thickness, in Softimage Units, of the border surrounding the fluid box in which the border avoidance will gradually become bigger and bigger.

Strength: the strength of the effect.

Velocity Forces: Enter the names of the objects, separated by ‘;’, that you want to use as velocity forces in the fluid box. You can have as many velocity forces as you want.

Vel.Noise: “Velocity Noise” lets you add some noise to the velocities contained in each cell of the fluid box. The resulting effect is similar to the velocity noise you can set in the property page of a particle type in XSI.

Note: there is more detailed information about this parameter set and its parameters in section 5.1.2.

You can experiment a little with the different values.

For example, you could set the value of “Vel.Noise_(Strength)” equal “50” to get the particles all excited. Or set the value of “Border_Avoidance (Thickness)” equal “10” and see how the particles squeezed together in the centre of the fluid box.

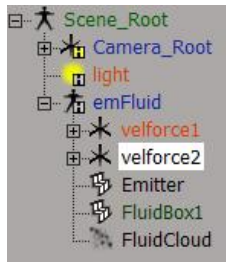
Or set the value of “Viscosity” equal “0.25”.

4.3. Tutorial 3: adding velocity forces

a) Load the scene “Tutorial_AddVelocityForces.scn”. Again, we have a fluid box and a velocity force.

b) Play back the scene. Particles are emitted left and right and the velocity force adds velocity that lets the left particles move, the right ones, though, are nearly not influenced.

c) Let’s add a second velocity force. Simply branch-select the velocity force “velforce1” and duplicate it:

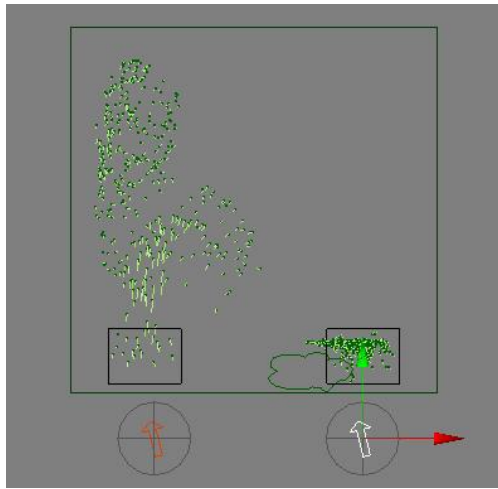


d) Now we need to tell *emFluid* about this new velocity force: open the property page of the fluid boxes custom parameter set “emFluidBoxParameters” and add the name of the new velocity force (“velforce2”) to the parameter “Velocity_Forces”. Make sure that the two names are separated by ‘;’:



Note: you can also enter the full name (in this case “emFluid.velforce2”). Do this, if you have several velocity forces with the same name but parented under different models.

e) Translate the new velocity source in positive x, so that it will influence the right particles:



f) Reset the particle cloud and play back the scene from the beginning. Both velocity forces now add velocity to the fluid box.

g) There is an animated object called “JustPassingBy”. Make it a velocity force, too, by simply adding its name to the parameter “Velocity_Forces” (note: won’t work with the demo version of *emFluid!*):



f) Reset the particle cloud and play back the scene from the beginning.

Note: any type of object can be used as a velocity force brush: nulls, meshes, lights, cameras, etc.

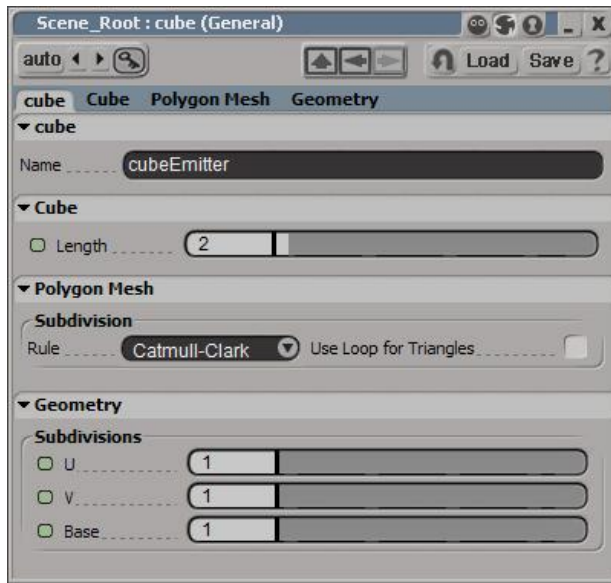
4.4. Tutorial 4: adding fluid boxes

a) Load the scene “EmptyScene.scn”.

b) Open the script editor and load the Visual Basic script “emFluid_CreateCompleteBasicSetup.vbs”.

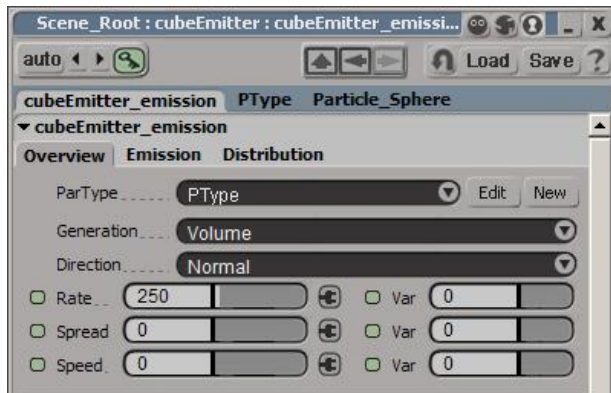
c) Run the script. Choose “Yes” when asked if you want a default animation. When the script is done, close the message box and the script editor.

d) First we are going to add a particle emitter to the cloud. Get a polymesh cube, set its length equal “2” and its name to “cubeEmitter”:



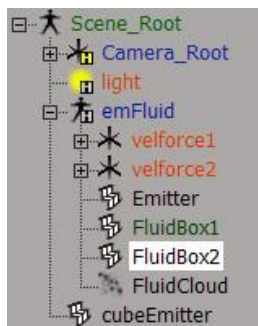
Close the property page and set the cubes x position equal “12” and y position equal “2”.

e) Select the cloud “FluidCloud” and call the function “Simulate->Modify->Particles->Add Emission”. Pick our freshly created cube (left click), then right click. A property page should pop up. In the emission settings set Generation to “Volume”, the rate equal “250” and the speed equal “0”:

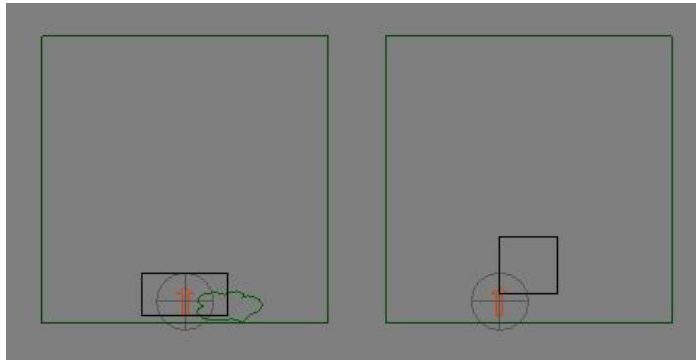


Close the property page.

f) Branch-select the velocity force “velforce1” and duplicate it, then select the fluid box “FluidBox1” and duplicate it, too:



Set the x position of “FluidBox2” equal “12” and the x position of “velforce2” equal “11”, so that you get something like this:

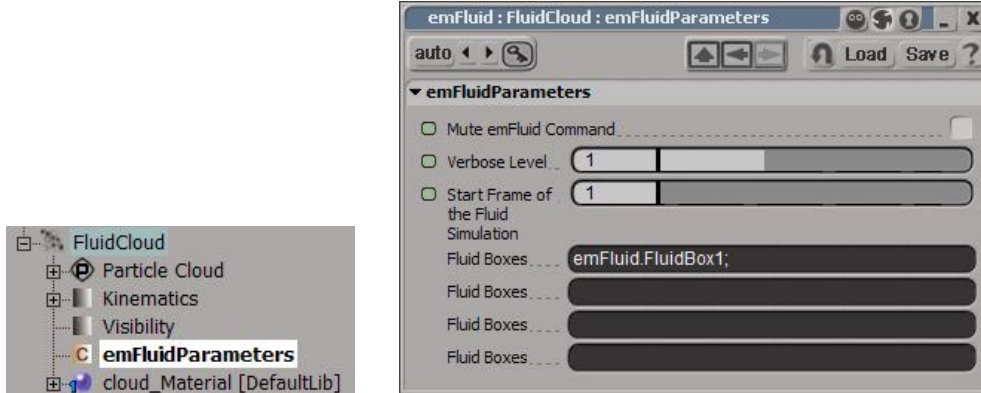


g) Now follow two important steps: we need to tell the cloud it has a new fluid box and we need to tell the new fluid box “FluidBox2” it shall use the velocity force “velforce2”.

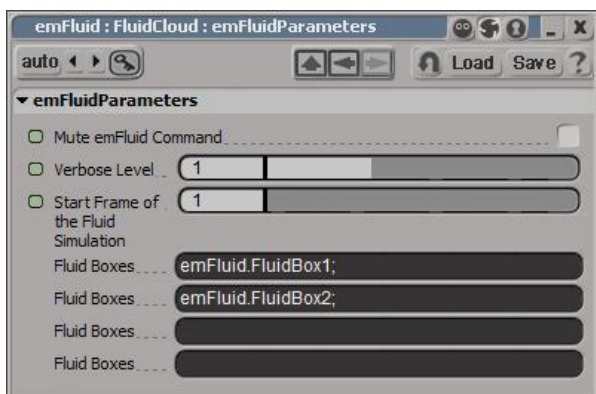
We start with the velocity force: open the property page of the custom parameter set of “FluidBox2” and set the parameter Velocity_Forces equal “velforce2”:



Close the property page and open the property page of the cloud “FluidCloud”:

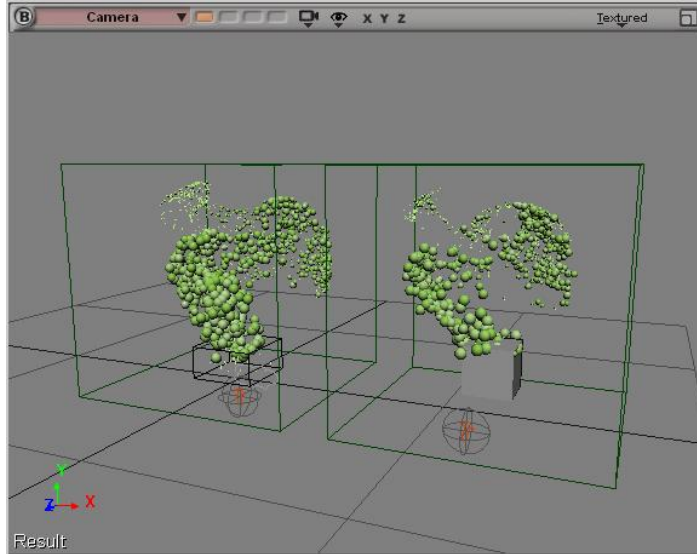


Enter the name of the new fluid box to one of the parameters “Fluid Boxes”:



Note: You must enter the full name (in our case “emFluid.FluidBox2”), and if more than one name is entered in a single “Fluid Boxes” parameter then they must be separated by ‘;’. Check section 5.1.1. for more information.

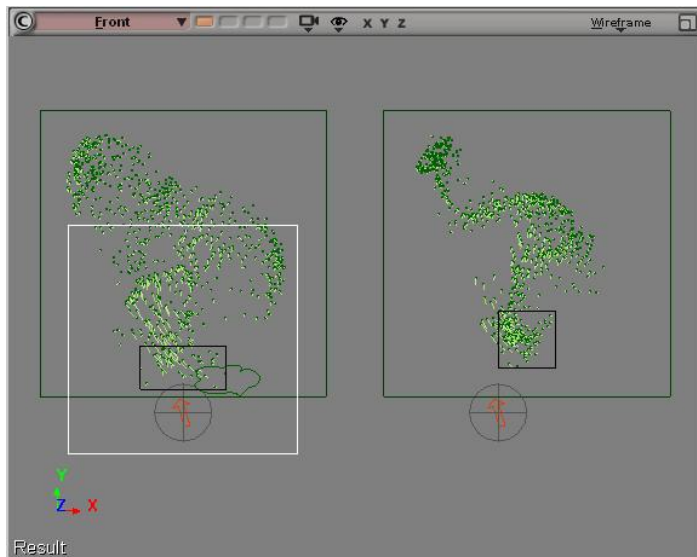
h) Close the property page and play back the scene. It should look about like this:



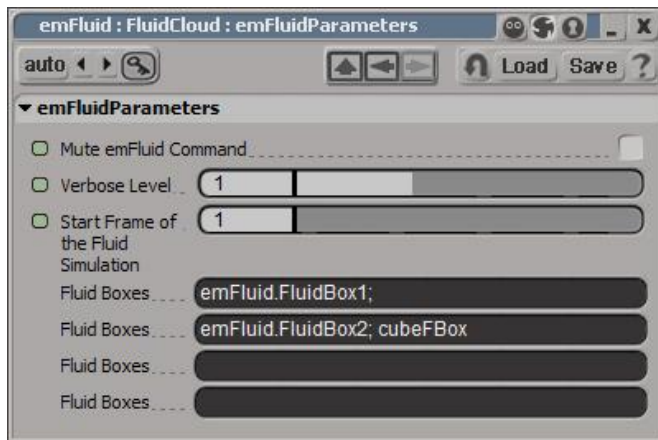
You can, of course, change some values of the second fluid box, e.g. add some velocity noise by setting “Vel.Noise_(Strength)” equal “10” (in the custom parameter set of “FluidBox2”), or change the animation of the velocity force “velforce2”.

Note: if you are using the demo version of *emFluid* then the rest of this tutorial won't work and you can end this tutorial here.

i) Let's add a third fluid box, but this time not by copying an existing one. Get a polymesh cube and set its name to “cubeFBox” and, in order to see something in the camera view, set its display mode to “wireframe”. Set its y position equal “2”. It intersects with the fluid box “FluidBox1”, but this is intended:



j) Open the property page of the cloud and add the name of the cube to the parameter “Fluid Boxes”:



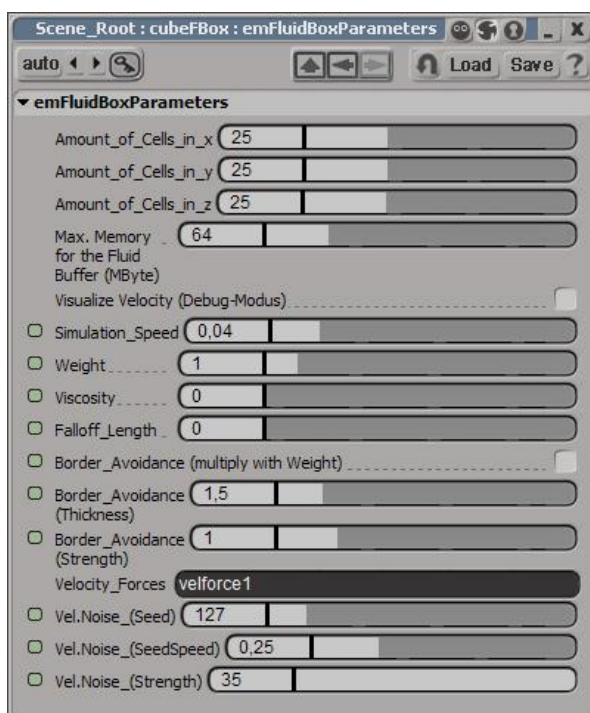
Close the property page.

k) The fluid box “cubeFBox” has no custom parameter set “emFluidBoxParameters”, but it is necessary for it to have one, else *emFluid* will simply ignore it.

There is a script for creating this parameter set: in the script editor open the VB script “emFluid_CreateParameterSet_FluidBox.vbs”.

Run the script. It will ask you to pick an object: pick “cubeFBox”. When the script is done the property page of the created custom parameter set will pop up and a message box will appear to inform you not to forget to add the object’s name to the parameter “Fluid Boxes”. We already did that in step i), so just close the message box.

In the property page of the created custom parameter set, set “Velocity_Forces” to “velforce1”, “Vel.Noise_(SeedSpeed)” equal “0.25” and “Vel.Noise_(Strength)” equal “35”:



k) Reset the particle cloud and play back the scene. You can observe how particles, which are in both left fluid boxes, are influenced by both left fluid boxes.
Hint: you can move fluid boxes around while the simulation is running. This sometimes gives interesting effects.

4.5. Tutorial 5: understanding “velocity force from geometry”

a) Load the scene “Tutorial_VelForceFromGeometry.scn”.

In this scene we have a polygon mesh grid called “Emitter_VForceGeo”. It is the particle emitter and, at the same time, a standard velocity force. This means that it adds velocity to the fluid box in direction of its local z axis, just like all the other velocity forces we had till now.

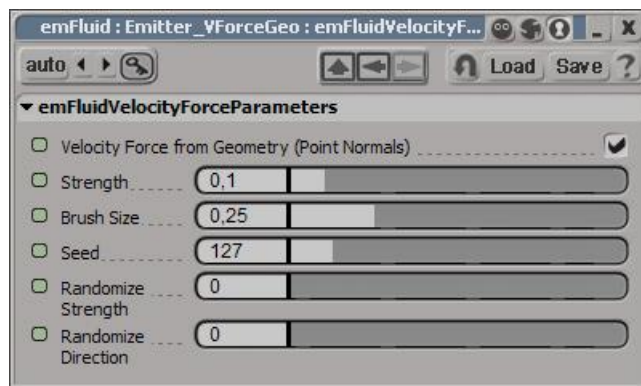
Note: if you are using the demo version of *emFluid* then you cannot do this tutorial, because “velocity forces by geometry” are only available in the full version.

b) Play back the scene. The simulation that looks pretty crappy.

c) We are now going to transform the velocity force into a “velocity force from geometry”. All there is to do, is to add a custom parameter set called “emFluidVelocityForceParameters” to the velocity force. We have a script that will do this for us: Open the Script Editor and load the script “emFluid_CreateParameterSet_VelocityForce.vbs”.

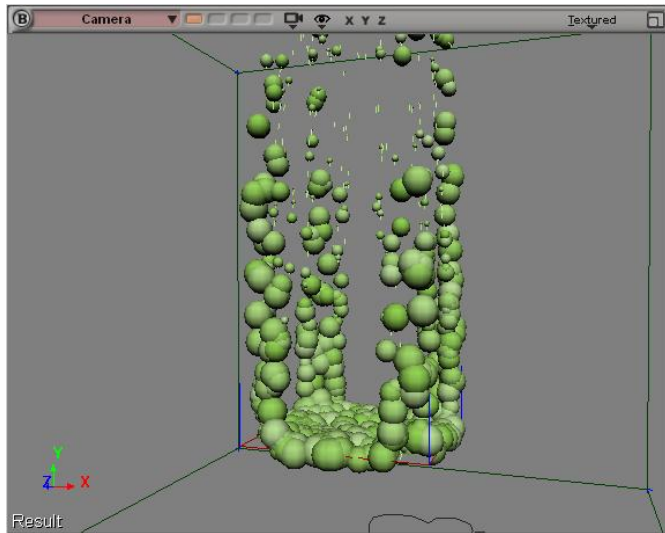
d) Run the script. It will ask you to pick an object: pick our velocity force “Emitter_VForceGeo”. When the script is done the property page of the created custom parameter set will pop up and a message box will appear to inform you not to forget to add the object’s name to the parameter “Velocity_Forces”. This is already the case, so just close the message box.

This is the newly created parameter set:

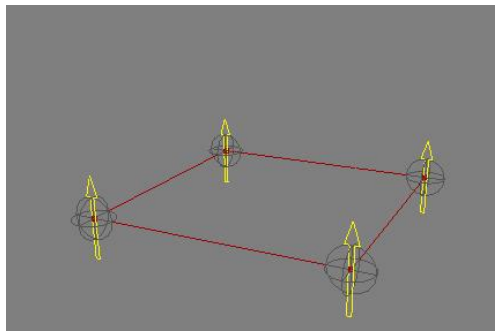
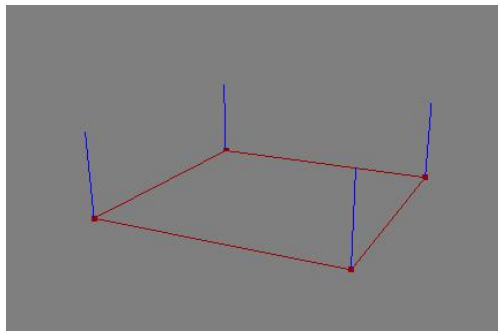


The first parameter called “Velocity Force from Geometry (Point Normals)” is checked, which means that our velocity force now is a “velocity force from geometry” (if we unchecked this parameter, we would have a velocity force just like in the previous tutorials, and the other parameters in this parameter set would be ignored).

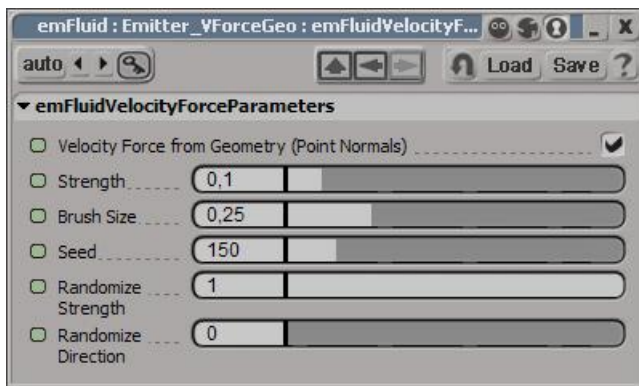
e) Reset the particle cloud and play back the scene from the beginning. It should look about like this:



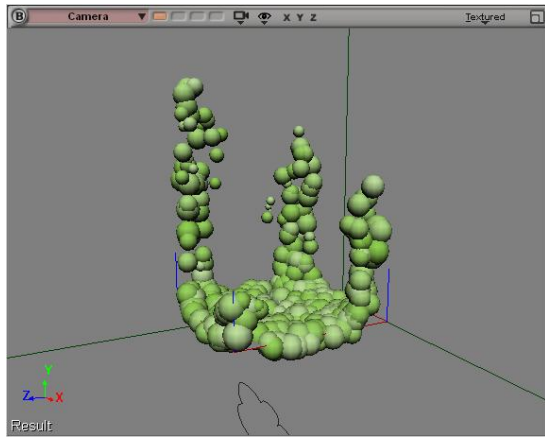
So what exactly does this velocity force do? It adds velocity to the fluid box, but, instead of taking the global position and local z axis of the velocity force object, it takes the polygon meshes points and point normals. It's as if you had a velocity force located at each point of the polygon mesh, pointing in the same direction as the points' normal vectors:



f) The strength of the added velocity is defined by the parameter "Strength" and is the same for all points of the polygon mesh. You can add some randomness to the strength via the parameter "Randomize Strength". Set the value of "Seed" equal "50" and the value of "Randomize Strength" equal "1":

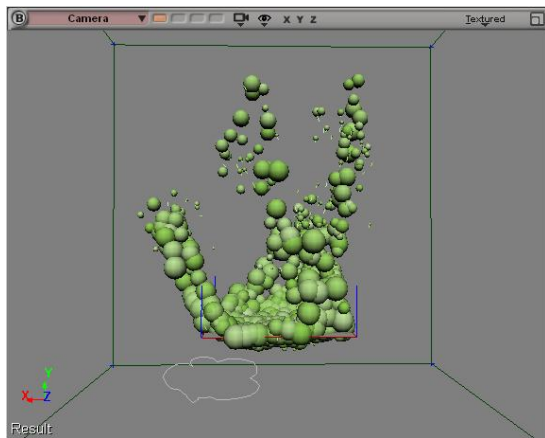


g) Reset the particle cloud and play back the scene from the beginning. You will see that some particle fountains are a little higher than others, because we added full randomness to the strength:



h) You can also add randomness to the direction of the velocity via the parameter “Randomize Direction”. Set the value of “Randomize Strength” equal “0” and the value of “Randomize Direction” equal “0.2”.

Reset the particle cloud and play back the scene from the beginning:



By changing the value of “Seed” you can “randomize the randomness”.

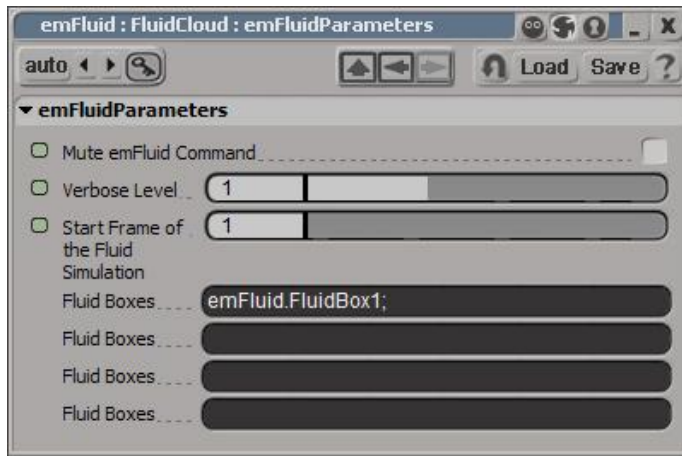
Take a look at the examples 9 and 10, which both use velocity forces from geometry.

5. Documentation

5.1. The custom parameter sets and their parameters

5.1.1. emFluidParameters

This custom parameter set belongs to the particle cloud.



The parameter set “emFluidParameters” has following parameters:

Mute emFluid command:

(Script name: **mute**)

If checked, then *emFluid* is muted and won't have any influence on the particles.

Verbose Level:

(Script name: **verbose**)

This sets the verbose level of *emFluid*. All output of *emFluid* can be found in the history log of the Script Editor.

Verbose level 0: disable verbose

Verbose level 1: minimum verbose. Only the calculation time and amount of particles is outputted, e.g. “'INFO : emFluid: total time (frame 21): 0.0365s (105 particles)”.

Verbose level 2: diverse calculation times are outputted.

Verbose level 3: heavy verbose.

Start Frame of the Fluid Simulation:

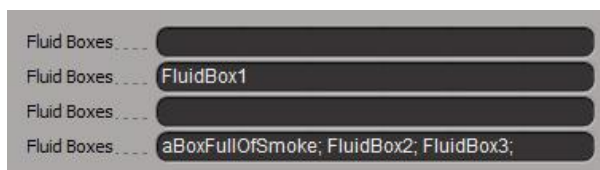
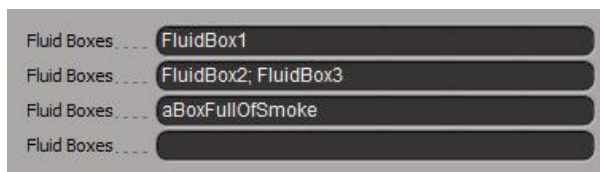
(Script name: **resetFBufFrame**)

This is the frame at which the fluid buffers are reset. This value should be identically with the first frame of the particle simulation.

Fluid Boxes:

(Script names: **fluidboxes1**, **fluidboxes2**, **fluidboxes3** and **fluidboxes4**)

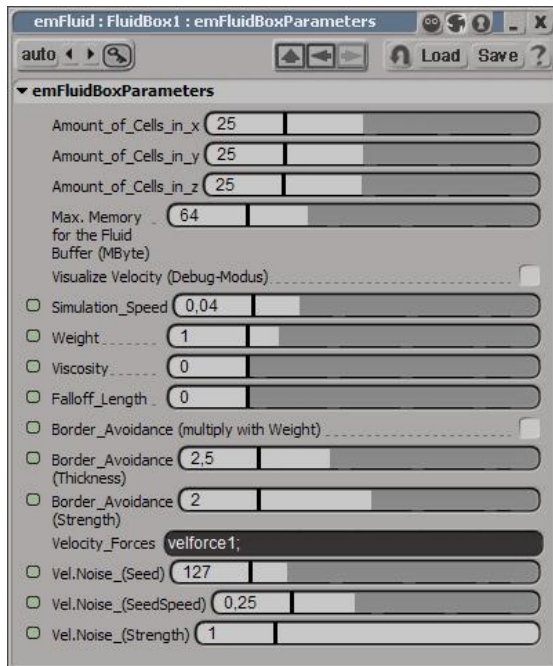
Here you enter the names of the objects (separated by ‘;’) that shall be used as fluid boxes. It makes no difference in which field the names are entered. The following two settings are therefore identical:



5.1.2. emFluidBoxParameters

This custom parameter set belongs to fluid boxes.

It can be created using the script “emFluid_CreateParameterSet_FluidBox.vbs”.



The parameter set “emFluidBoxParameters” has following parameters:

Amount of cells in x/y/z:

(Script names: N_x , N_y , N_z)

The volume (or space) given by the fluid box is divided into voxels. With these three parameters you define the subdivision in x, y and z of the space and therefore the number of voxels. Greater values require more memory and longer calculations.

Note: in order for *emFluid* to produce correct results, the global size of the fluid box should match the subdivisions of the space, e.g. if your subdivisions are $N_x=20$, $N_y=30$ and $N_z=20$ then the global size of the fluid box should be 1.5 times greater in y than in x and z, for example 5 in x, 7.5 in y and 5 in z.

Max. Memory for the Fluid Buffer (Mbyte):

(Script name: **maxMB**)

Defines the upper limit of memory, in Megabytes, the fluid buffers may use. In case they should require more memory, an error message will be displayed.

This parameter is a security, so that you won't allocate too much memory by mistake. Here some examples of the memory usage for different values of “Amount_of_cells_in_x/y/z”:

25x25x25: 0.0357 MByte

64x64x64: 6 MByte

128x128x128: 48 MByte

256x256x256: 384 MByte

In most cases, though, values between 20 and 50 should be sufficient.

Visualize Velocity (Debug-Modus):

(Script name: **debugmod**)

When on, the particles are positioned in the centres of the voxels and have the velocity of the voxel.

Simulation_Speed:

(Script name: **speed**)

This defines the speed of the fluid simulation.

To produce correct results set this value to 1 divided by the frame rate of your scene, e.g. if your scene's frame rate is 25, then set this value equal $1/25 = 0.04$, if your scene's frame rate is 30, then set the value equal $1/30 = 0.0333$, etc.

Hint: if you want to speed up the particles in the fluid box, increase the value of **Weight** (see next parameter) instead of changing the value of **Speed**.

Weight:

(Script name: **weight**)

This defines how strong the fluid box will influence the particles that are contained in it. A value of 1 means full influence, a value of 0 means no influence at all.

Use values greater than 1 if you want the particles in the fluid box to move faster.

Use values smaller than 1, if you want the particles to be less influenced by the fluid box. This is especially recommended when using XSI forces.

You can also use negative values for a sort of reverse playback effect.

Viscosity:

(Script name: **visc**)

This defines the viscosity (the "liquidness"). Higher values give a more syrup – like effect.

Falloff_Length:

(Script name: **falloff**)

Defines the thickness, in Softimage Units, of a border surrounding the fluid box in which the influence of the fluid box will gradually become smaller and smaller. This means: the nearer a particle gets to the outer limit of the fluid box, the less influence the fluid box has on the particle. This also means particles can leave the fluid box.

This is useful when combining several fluid boxes.

Border_Avoidance:

(Script names: **bordMulWeight**, **bordThick** and **bordStrength**)

Border avoidance means: the nearer a particle gets to the outer limit of the fluid box, the more it will want to get away from the border.

Multiply with Weight: if checked, then the amount of border avoidance is multiplied with the parameter "Weight".

Thickness: the thickness, in Softimage Units, of the border surrounding the fluid box in which the border avoidance will gradually become bigger and bigger.

Strength: the strength of the effect.

Velocity_Forces:

(Script name: **velforces**)

Enter the names of the objects (separated by ';') that you want to use as velocity forces in the fluid box, e.g. if you want to use the nulls "myVforce1" and "Model.Velforce" as velocity forces then enter "myVforce1; Model.Velforce".

You can have as many velocity forces as you want.

Vel.Noise:

(Script name: **vnoise_seedStart**, **vnoise_seedSpeed**, and **vnoise_strength**)

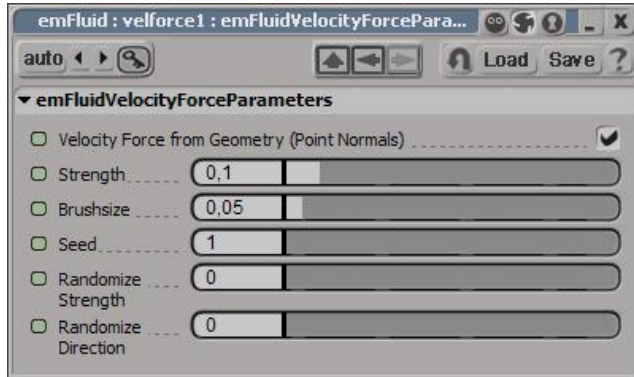
“Velocity Noise” lets you add some noise to the velocities contained in each cell of the fluid box. The resulting effect is similar to the velocity noise you can set in the property page of a particle type in XSI.

5.1.3. emFluidVelocityForceParameters

This custom parameter set belongs to fluid boxes.

It can be created using the script “emFluid_CreateParameterSet_VelocityForce.vbs”.

Tip: the tutorial 5 is highly recommended to fully understand velocity forces that own this parameter set, so called “velocity forces from geometry”.



The parameter set “emFluidVelocityForceParameters” has following parameters:

Velocity Force from Geometry (Point Normals):

(Script name: **useNormals**)

When this parameter is checked, then the velocity force will be emitted from the points (of a polymesh) in direction of the points’ normals. The velocity force becomes a “velocity forces from geometry”.

When this parameter is unchecked, then all other parameters are ignored.

Strength:

(Script name: **strength**)

Sets the strength of the velocity that is emitted from the points.

Brush Size:

(Script name: **brushsize**)

Defines the virtual brush size for a point.

Seed:

(Script name: **rndSeed**)

Defines the random seed. Animate this parameter if you want to change the randomness strength and direction, e.g. by adding the expression “Fc”, the randomness would change each frame.

Randomize Strength:

(Script name: **rndScl**)

The amount of randomness for the velocity that is emitted from the points.

A value of 0 means no randomness, a value of 1 means that the emitted velocity will be between zero and “Strength” (parameter described above).

Randomize Direction:

(Script name: **rndDir**)

The amount of randomness for the direction of the velocity that is emitted from the points.

A value of 0 means the velocities' directions follows exactly the points' normals, a value of 1 means the directions are rotated randomly in x, y and z by 180 degrees.

5.2. The Scripts (and what they're there for)

“emFluid CreateCompleteBasicSetup.vbs”

This script will create a complete basic setup to animate particles with *emFluid*.

It creates the following setup:

- a particle cloud with its custom parameter set “emFluidParameters”
- a particle type with a scripted event that calls the command *emFluid*
- a particle emitter
- a fluid box with its custom parameter set “emFluidBoxParameters”
- a velocity force for the fluid box

All above objects are parented under a model and you can directly play back the scene.

Note: the script will prompt you to add –or not- a default animation to the velocity force. The animation consists of three expressions that control the velocity force's local x and z rotation, as well as the local y position (also see tutorial 1 in section 4.1.).

“emFluid CreateParameterSet FluidBox.vbs”

This script will ask you to pick an object and then add a custom parameter set called “emFluidBoxParameters” to it. The parameter set contains the parameters for controlling the fluid box, e.g. memory limits, weight and viscosity.

For a complete list and description of the parameters go to section 5.1.2.

Note: an object that shall be used as a fluid box must be listed in the parameter “Fluid Boxes” of the custom parameter set of the cloud.

“emFluid CreateParameterSet VelocityForce.vbs”

This script will ask you to pick an object and then add a custom parameter set called “emFluidVelocityForceParameters” to it. This parameter set contains the parameters for controlling geometry velocity forces, e.g. Strength and Randomness.

For a complete list and description of the parameters go to section 5.1.3.

Note: an object that shall be used as a velocity force must be listed in the parameter “Velocity_Forces” of the custom parameter set of the fluid box (or fluid boxes).

“emFluid PEvent.vbs”

This VB script contains the code that is executed by a particle event in the cloud. It reads the values of the custom parameter sets of the cloud, the fluid boxes and the velocity forces. It will then call the command *emFluid* that will calculate the behaviour of the particles that are within a fluid box.

6. Trouble shooting

Problem: XSI outputs the following error:

```
" 'ERROR : - [line 177 in ..\Plugins\emFluid_PEvent.vbs]"
```

Cause: XSI cannot find the dll "emFluid.dll".

Solution: copy the file "emFluid.dll" into one (and only one!) of the following paths:

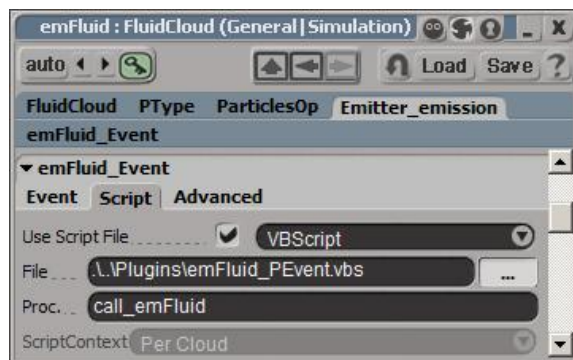
- The folder "Application\Plugins" that is located wherever you installed XSI, e.g. "C:\Softimage\XSI_5.11\Application\Plugins".
- The folder "Application\Plugins" that is located in XSI's user path, e.g. "C:\users\Administrator\Softimage\XSI_5.11\Application\Plugins"

Problem: XSI outputs the following error:

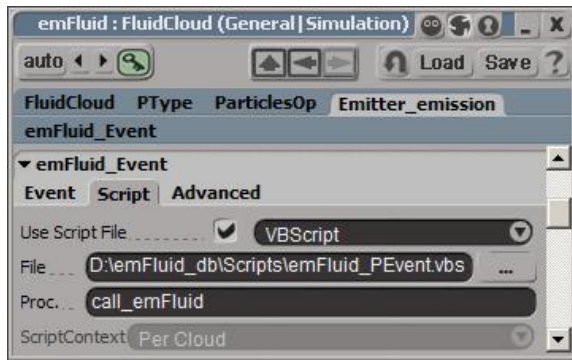
```
''ERROR : 2000 - File not found - ..\Plugins\emFluid_PEvent.vbs"
```

Cause: XSI cannot find the VB script "emFluid_PEvent.vbs", because the path is incorrect.

Solution: when a scripted particle event uses a file (=> the parameter "Use Script File" is checked), then the complete path and filename of the script file must be specified. In the example scenes, the tutorial scenes as well as in scenes generated with the script "emFluid_CreateCompleteBasicSetup.vbs" the path is not absolute, (as it should be) but relative. It may happen that XSI's current path changes, and this means that the relative path also changes, and that means XSI can't find the VB script file anymore. To fix this problem, simply relocate the file "emFluid_PEvent.vbs". You can use the file "emFluid_PEvent.vbs" in the folder "Scripts" of the database "emFluid_db":



Filename with *relative* path (can cause problems):
Relocate the file "emFluid_PEvent.vbs" by clicking on the button "..." or by entering the path and filename directly.



After relocation of the file “emFluid_PEvent.vbs”: filename with *absolute* path, which should not cause any problems.

Problem: when I make changes to a fluid box or a velocity force, XSI doesn’t automatically recalculate the particles.

Cause: all the emFluid - objects are only connected via scripts and names. This means, that XSI doesn’t know that there is a link between, for example, a fluid box and the particles. *emFluid* is only a command, not an operator (maybe in the future, who knows?).

Solution: you must manually tell XSI to recalculate the simulation by clicking on “Simulate” in the ParticleOp tab.

Problem: a fluid box doesn’t seem to influence the particles.

Possible causes: *emFluid* doesn’t know anything about the fluid box *and/or* the fluid box has no velocity forces *and/or* the weight of the fluid box is equal 0.

Solution: add the name of the fluid box to the custom parameter set of the particle cloud (parameter “Fluid Boxes”) *and/or* add velocity forces (or velocity noise) to the fluid box *and/or* set the weight of the fluid box smaller than or greater than 0.

Problem: a fluid box somehow influences the particles in a wrong way.

Possible causes: the weight of the fluid box is smaller than 0 or much bigger than 1 *and/or* the global x, y and z scaling of the fluid box don’t go well with the values “Amount_of_cells_in_x/y/z” of the parameter set “emFluidBoxParameters” *and/or* the value of the parameter “Simulation Speed” doesn’t match the scenes frame rate.

Solution: correct the weight value *and/or* check section 5.1.2. concerning correct values for “Amount_of_cells_in_x/y/z” and “Simulation Speed”.

Problem: a rotated fluid box somehow won’t do what I want it to do.

Cause: the rotation.

Solution: the global translation and scaling values of a fluid box can have any value, but the **global** rotation should always be 0 in x, y and z. Why is this? *emFluid* calculates the particles behaviour in the

global bounding box of a fluid box. This means, that any rotation only affects the size of the global bounding box, but not its rotation, because global bounding boxes don't have rotation values unequal 0.

Problem: *emFluid* is so damn slow I could cry.

Possible causes: one or more fluid boxes probably have very large values for "Amount_of_cells_in_x/y/z". As space is 3-dimensional, calculation time increases rapidly when you enter large values, e.g. if the parameters "Amount_of_cells_in_x/y/z" of a fluid box are equal "25 x 25 x 25" and it takes 0.1 seconds to calculate the whole stuff, then it would take about $4*4*4 = 64$ times more time to calculate a fluid box of "100 x 100 x 100" which would be 6.4 seconds.

Solution: try smaller values for "Amount_of_cells_in_x/y/z".

Problem: after a long session with *emFluid* the memory XSI requires is gigantic and the computer starts swapping memory all the time, slowing everything down.

Possible causes: I spend much time looking for memory leaks in *emFluid*, but simply couldn't find any. Then I discovered that when you use scripted Particle Events, XSI allocates a certain amount of memory (depending on the amount of particles) per frame, but unfortunately doesn't free this memory. So even if your script is something ultra simple like e.g. "a=1", you will still get some memory problems after a longer session.

This may also depend on what Version of XSI you are working with (5.xx or 6.0x).

Solution: It takes quite a while before the memory is full, so it is not that a big problem. But still, if it occurs, then simply close XSI and restart XSI.

Problem: the particles in my cloud somehow are influenced by fluid boxes without being in them.

Possible causes: the particle cloud object is scaled, rotated or translated.

Solution: reset the translation, rotation and scaling.

Problem: some particles leave the fluid box even though they shouldn't.

Possible cause: some particles have so much velocity that they manage to break through.

Solution: make the fluid box a XSI obstacle for the particles (set the obstacle's parameter "push length" equal a value greater than zero), then all particles should stay inside, no matter how fast they are. Another solution would be to increase the values of the fluid boxes border avoidance.

Email Contact: if you should have any questions or comments regarding *emFluid* you can contact us via

`xsi[at]skylife[dot]de`

We would also be glad to receive some of your *emFluid* work examples.

9. Thanks

This plug in is based on Jos Stam's paper "Real-Time Fluid Dynamics for Games" and I would like to thank him for his great work that has been inspiring programmers for over ten years!

I would like to thank Oliver Weingarten (www.pixelpanic.de), the friend who motivated me to code this plug in and who did a great beta testing job, Christian Halten (www.skylife.de), the friend who supported me with this and so many other projects, and, last but not least, my beta testers Achim Dietze, Andreas "andialias" Schulz and Stephan Heitz. Thanks, guys!

And don't forget to visit the SKYLIFE ONLINE SHOP from time to time to check for new products:

www.shop.skylife.de