Particle Universe Manual

- Editor -

Author: Henry van Merode
Version 1.6
www.fxpression.com

Introduction

Particle Universe is a complete system to create visually stunning particle systems for Ogre[1]-powered video games and video editing. The package consists of an editor for creation of particle systems using a visual editing system, and a runtime plugin for in-game usage of the created particle system scripts. Although the editor runs on the Windows™ platform, the plugin itself is not specifically created for Windows™. The source code (C++) of the plugin is included in the package[2].

Besides usage for video games, the editor can also be used to create special effects for video editing. It is possible to export a particle system to a series of image files, with or without alpha channel or background colour, or export to a video format; avi, wmv and swf are supported. The image files or video files can be imported into a video editing application.

This manual is used as an introduction for the Particle Universe editor. By means of a few lessons, its capabilities are explained.

[1] Ogre is a multiplatform rendering system that is widely used in commercial video games. See www.ogre3d.org

[2] Particle Universe by default runs on Windows™. Other platforms are not supported out-of–the-box.

Particle Universe Editor source setup

The Particle Universe Editor is an executable (in case of Windows™) that runs as an individual application. The Visual Studio solution files (.sln) and all C++ code are included in the package.
The Particle Universe Editor package comes with both a binary version of the Particle Universe Editor and a C++ source version. If you only want to use the Particle Universe Editor and don't want to make changes to the source, you may skip this chapter.

If you want to make changes to the source, or just want to compile the editor, please proceed with the following steps.

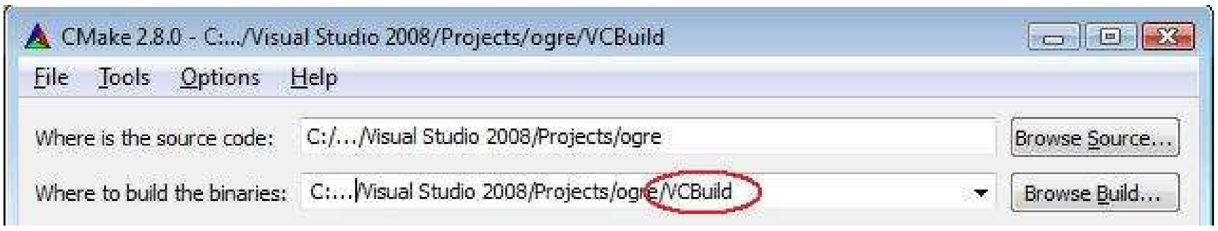
Setup editor

Install the Particle Universe Editor package by starting the Window installer. This results in the creation of the following directory structure:

- Particle Universe
  - Particle Universe Editor
  - VCBuild
    - Particle Universe Editor
    - Plugins
    - Samples
    - Tools

The top *Particle Universe Editor* directory contains the executable of the editor. The *VCBuild* directory contains the source and addional files to make everything work. This includes both the source of the Particle Universe plugin and the Particle Universe Editor.

When building the Ogre components, assume that you selected one of the Visual Studio compilers in the CMake dialog and generated the necessary files to the *VCBuild* directory as shown in the CMake dialog below.



CMake 2.8.0 - C:.../Visual Studio 2008/Projects/ogre/VCBuild

Where is the source code: C:/.../Visual Studio 2008/Projects/ogre

Where to build the binaries: C:.../Visual Studio 2008/Projects/ogre/VCBuild

Of course this may be a complete different directory, but for convenience it is assumed that the directory is named *VCBuild* (and so is the directory in the Particle Universe Editor package).

Copy all subdirectories and files of *VCBuild* to the *VCBuild* directory (or an alternative) of Ogre, that was generated with CMake (which contains all the Ogre code). All Particle Universe files are copied to the Ogre directory strucure.

Dependencies

The editor depends on:

- Ogre3D (editor version 1.6 is powered by Ogre3d V1.8.1)
- wxWidgets V2.8.9 with Unicode support, see www.wxwidgets.org.
- wxPropertyGrid V1.4.3.
- Particle Universe Plugin DLL.

Compile the three packages. For convenience, a package containing a stripped down version of wxWidgets and wxPropertyGrid can be downloaded from www.fxpression.com.
Make sure that you set an environment variable, called **WXWIN**. This refers to the wxWidgets headers and lib files.

Compile

The package contains a ***ParticleUniverseEditor.sln*** file. It has a reference to environment variable **WXWIN**, but there is no Ogre reference variable, so all paths to Ogre files are relative (relative paths from the VCBuild subdirectory to the Ogre root directory).

Open ***ParticleUniverseEditor.sln*** in Visual Studio and build all.

Run

Before you run the editor, make sure that the file

`VCBuild\Particle Universe Editor\`***`ParticleUniverseEditor.exe`***

is build.

Copy the following files (the ones that are build by the dependencies):

- ***OgreMain.dll***
- ***ParticleUniverse.dll***
- ***RenderSystem_Direct3D9.dll***

to the directory
`VCBuild\Particle Universe Editor\`

Do the same with the files:

- ***OgreMain_d.dll***
- ***ParticleUniverse_d.dll***

- ***RenderSystem_Direct3D9_d.dll***

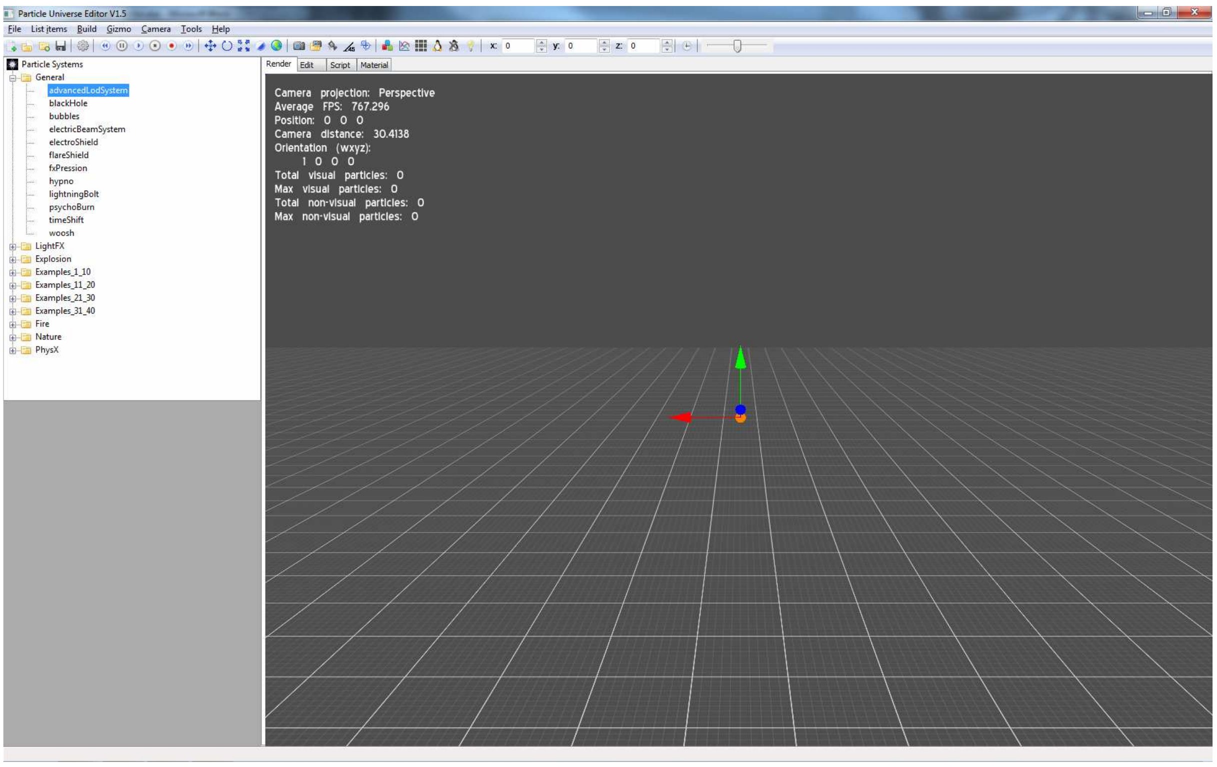
if you want to debug the editor.

Now start ***ParticleUniverseEditor.exe***

Quick Tour

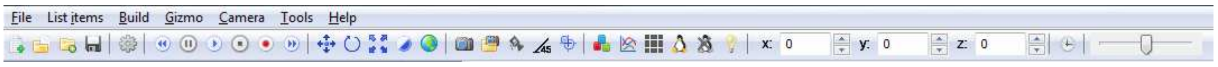
Editor

After starting the editor, a screen that resembles the figure below, is presented. By default, a number of precreated particle system scripts are included in the treelist on the left.



```
Camera projection: Perspective
Average FPS: 767.296
Position: 0 0 0
Camera distance: 30.4138
Orientation (wxyz):
    1 0 0 0
Total visual particles: 0
Max visual particles: 0
Total non-visual particles: 0
Max non-visual particles: 0
```

Menu



The top menu bar includes several icons for fast access to the editor functions. From left to right these functions are:

New particle system

Create a new ‘empty’ particle system.

Load particle system

Load a Particle Universe script (*pu or *pua) file[3]. The script file may contain multiple particle system definitions, which are all added to the list, after loading. If a particle system is already in the list, the particle system with the same name is still loaded, but with the prefix 'CopyOf' in front of the name.

Add resource location

Makes it possible to select a complete directory. All resources in this directory that are known to the editor, are loaded. This not only includes *.pu or *.pua files, but also

- material files (*.material), which refer to a texture.
- mesh (*.mesh) files, which represent 3D objects.
- Textures, the actual image files.

Resource locations that are added by the 'Add resource location' function are stored and used when the editor is restarted. The menu contains a configuration screen that makes it possible to deselect the resource locations again. This function can be found in 'Tools' > 'Options'.

When the editor is started, also some default resource locations are loaded. These resource locations contain the examples, the mediapack and the core media files. The default resource locations are included in a file called ***resources.cfg***.

Save particle system

Saves the currently selected particle system script.

Compile

This function can only be used if the Script-tab is selected (see later in this chapter). A changed script can be verified by pressing the compile icon (or the F5 function key).

The next 6 icons are the 'play' icons. They represents (from left to right):

- **Rewind** – Select the previous particle system in the list.
- **Pause**– Pause a playing particle system.
- **Play** – Start the current particle system
- **Stop** – Stop a playing particle system.

[3] A *.pu file is a regular particle universe script file. A *.pua file is an alias file that includes predefined pieces of script, which are reusable in other *pu files.

Record – Record the current particle system; this saves a video file or a series of image files of a 'playing' particle system.
Wind - Select the next particle system in the list.

Move

Represents the move gizmo on the Render tab. If this icon is selected, any object selected in the rendered scene can be moved across the scene.

Rotate

Represents the rotation gizmo on the Render tab. If this icon is selected, any object selected in the rendered scene can be rotated. This only applies to objects that can be rotated. I.e. a light cannot be rotated.

Scale

Represents the scale gizmo on the Render tab. If this icon is selected, any object selected in the rendered scene can be scaled. In practice, this only applies to a mesh added to the scene.

Particle systems can also be scaled, but not on the Render tab. Particle systems must be scaled on the Edit tab.

Toggle gizmo

Clicking on this icon enables/disables the gizmo's on the render window.

World/object space gizmo

/

Move and rotate gizmo's can act in world- or object space. If the worldspace icon is visible, movement and rotation is done by means of the x,y- and z-axis of the scene. If the objectspace icon is visible, movement and rotation is done by means of the x,y- and z-axis of the selected object.

Camera reset

The camera position can be saved. Pressing this icon restores the original (saved) camera position.

Camera save

Saves the camera position.

Perspective/orthographic camera

/

Version 1.5 introduced two camera projection types. Toggle between the two camera modes is done by pressing this icon.

45 degree / perpendicular orthographic camera view

45 / 90

In the orthographic camera mode, two possible views can be selected. The 45 degree view represents a topdown view, while the perpendicular view displays the scene from one of the selected X, Y or Z angles (these angles can be selected with the gizmo showing at the top-right of the screen).

Centre object

The selected object (mesh, particle system or light) is centred if this icon is pressed.

Background colour

Makes it possible to change the colour of the render window.

Toggle statistics

Clicking on this icon enables/disables the statistics overlay. It displays the:

- Camera projection type (Perspective or Orthographic).
- Average FPS (Frames per second).

- Distance between the camera and the selected object.
- Orientation of the selected object (in Quaternion values).
- Total number of emitted visual particles.
- Maximum number of emitted visual particles when the particle system was started.

Toggle grid plane

Clicking on this icon enables/disables the grid plane in the scene.

Add mesh

Adds a mesh to the scene. A listbox is displayed with all the meshes that are included in the resource locations. If the Render-tab is set, a mesh can be added to the scene.

*There is an alternative method to add a mesh to the scene. Just drag and drop a *.mesh file from the Windows file explorer to the Rendertab loads the mesh into the scene. Any reference to materials, textures, skeleton files and shaders displays a directory dialog from where the location of these resources can be selected.*

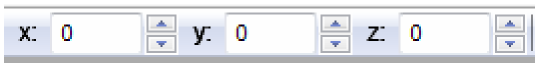
Remove mesh

Removes a mesh from the scene.

Toggle light

Enables/disables a point light. This can be used for mesh/particle materials for which lighting is enabled.

Euler rotation


x: 0 y: 0 z: 0

A mesh or particle system can be rotated by means of the rotation gizmo, but it is also possible by means of entering the rotation (in degrees) into these three fields. Note, that the order of rotation gives different results:

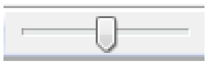
- If the X-value is entered last, the rotation order is: XYZ

- If the Y-value is entered last, the rotation order is: YXZ
- If the Z-value is entered last, the rotation order is: ZXY

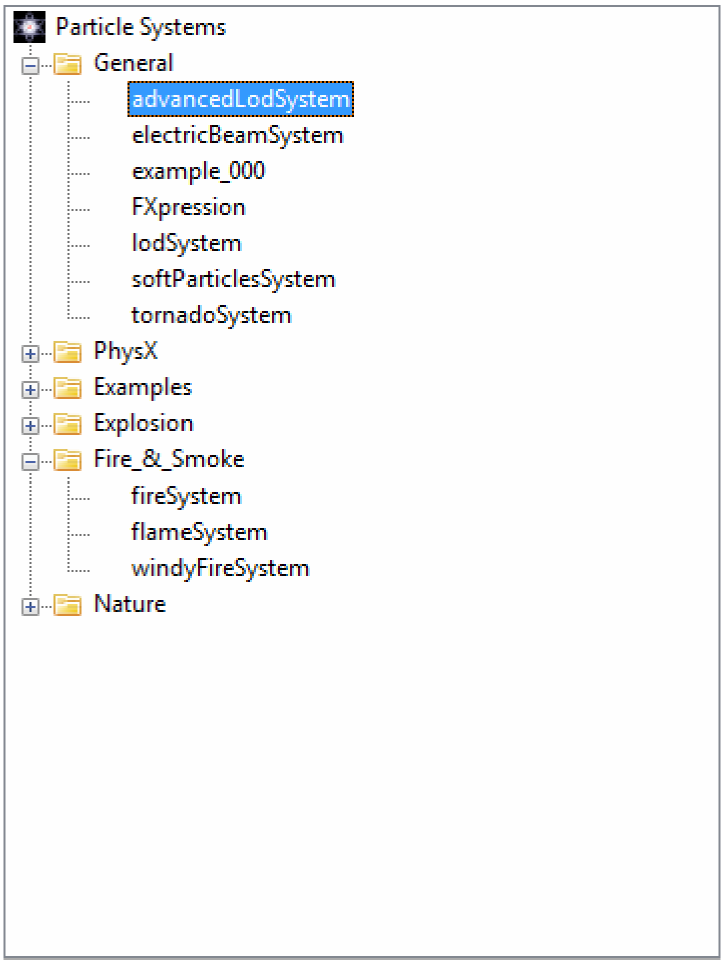
Time reset

This resets the time of the ‘time’ slider.

Time slider

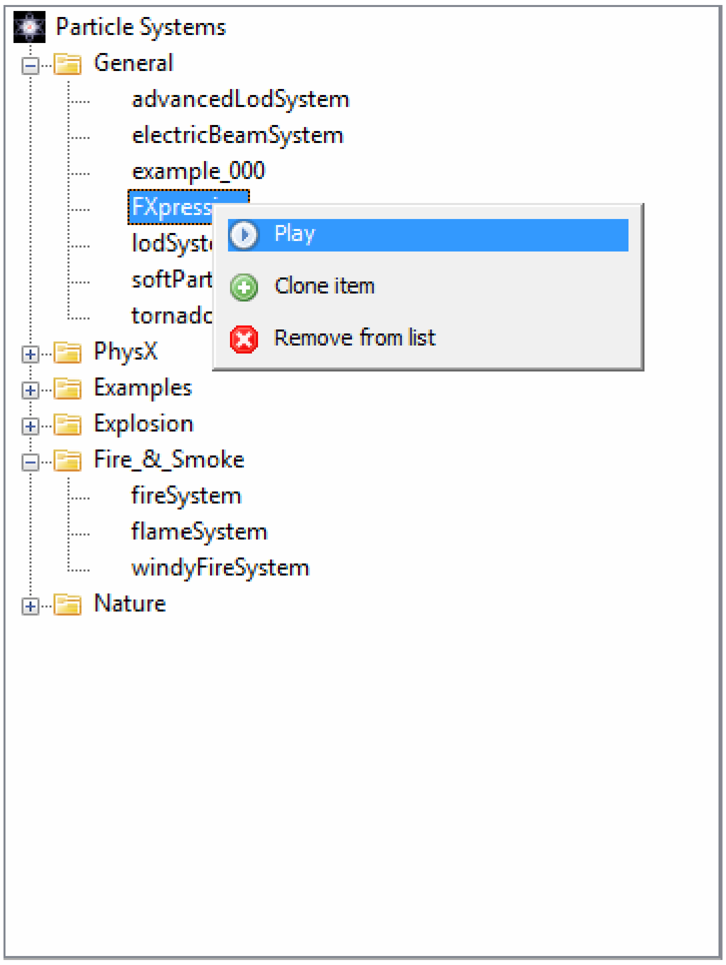

With the time slider, a running particle system and animated mesh can be speeded up or slowed down, so analysing the behaviour of the particle system during runtime becomes easier. The ‘Time reset’ button sets the time to its original value.

Particle System Treelist



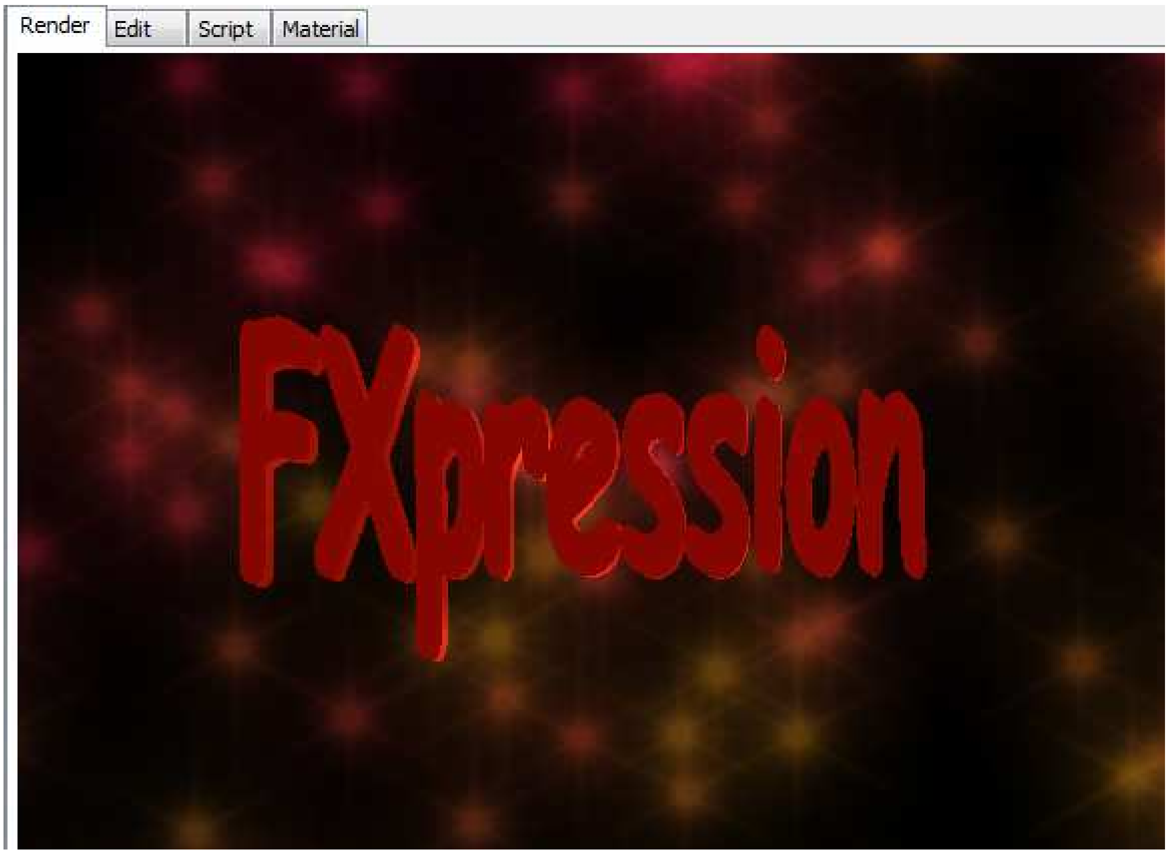
On the left of the screen, all loaded particle systems are displayed in a treelist. The treelist consists of categories; a category can be assigned to a particle system. The treelist currently does not allow drag and drop particle systems from one category to another, but by means of the Edit- or Script-tab it is easy to assign a particle system to another category. The highlighted entry is the active particle system.

If the mouse cursor is in the treelist, clicking the right mouse button displays a small context menu.



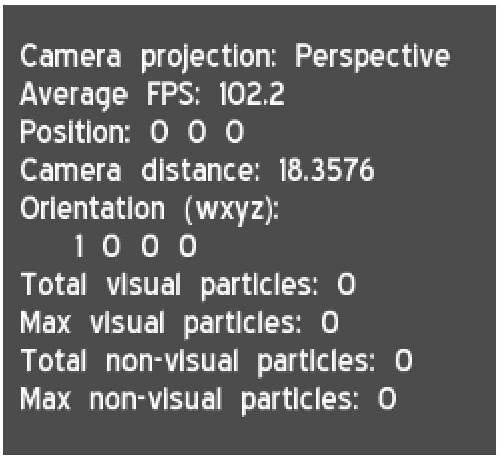
Play – starts the particle system.
Clone item – makes a copy of the active particle system, with the prefix ‘CopyOf’.
Remove from list – removes the particle system from the list, but doesn’t delete it from any file in which the particle system script is included.

Render-tab



The Render-tab is the main render window in which the current particle system can be viewed. The camera can be rotated and zoomed in- and out, to have a different view on a particle system or a mesh. Rotate the camera around the selected object by dragging the mouse with the right mouse key pressed. The scrollwheel is used to zoom in- and out.

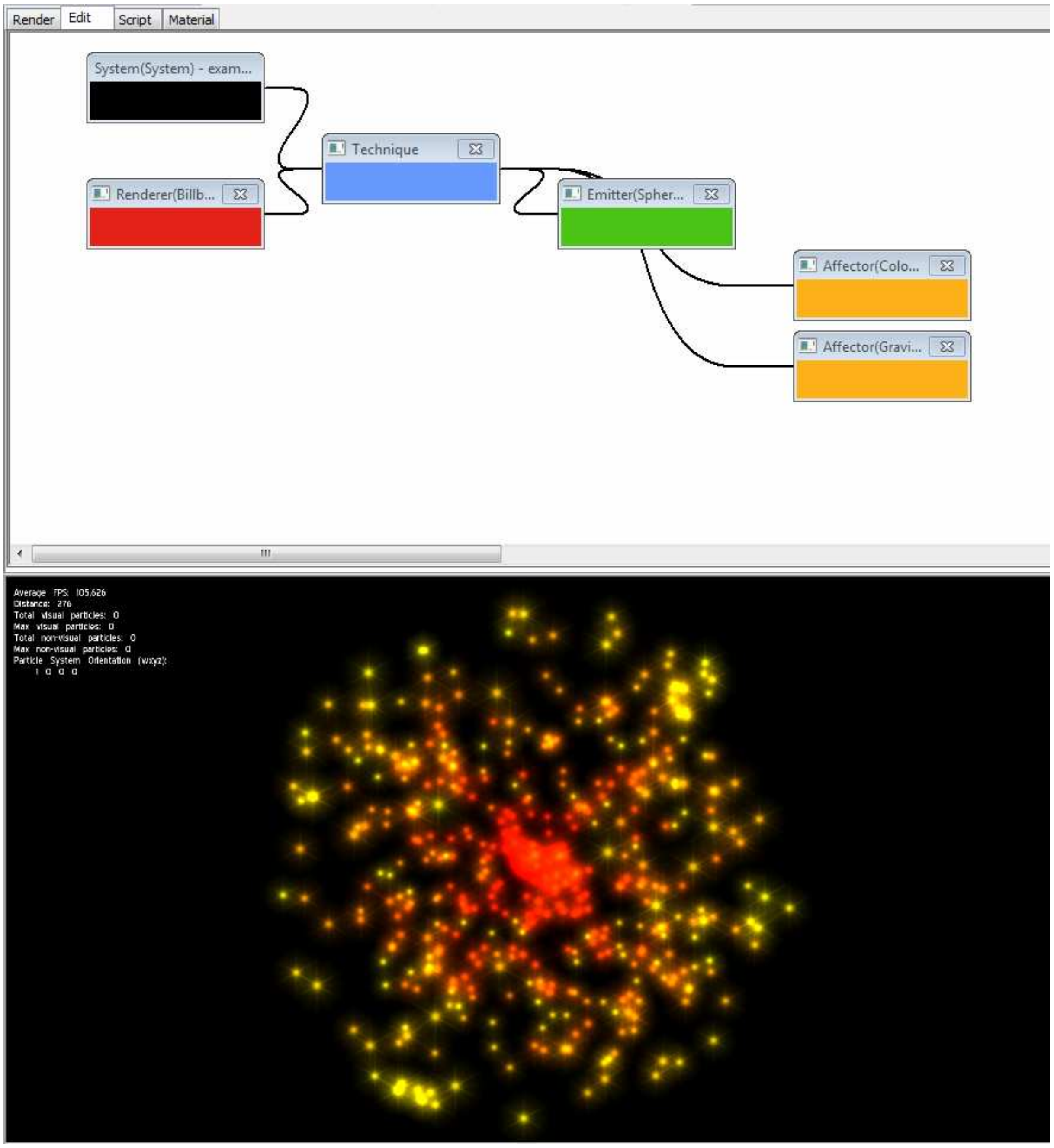
Statistics



```
Camera projection: Perspective
Average FPS: 102.2
Position: 0 0 0
Camera distance: 18.3576
Orientation (wxyz):
    1 0 0 0
Total visual particles: 0
Max visual particles: 0
Total non-visual particles: 0
Max non-visual particles: 0
```

Statistics are displayed on the Render-tab, which can be set on or off in the menubar.

Edit-tab



```
Average FPS: 105.626
Distance: 276
Total visual particles: 0
Max visual particles: 0
Total non-visual particles: 0
Max non-visual particles: 0
Particle System Orientation (wxyz):
     1 0 0 0
```

The Edit-tab will be used most of the time (unless someone finds it more comfortable to edit a script). The Edit-tab consists of two parts. The upper part is used to visually connect all components of which a particle system consists. The lower part is a renderwindow in which the changes are applied in real-time.

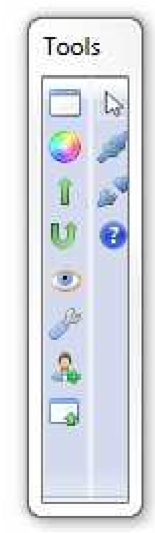
Both parts of the window are equal in size. It is not possible to change size ratio dynamically (yet), but it can be changed in the configuration file. This file – named pued.cfg – contains the setting 'edit_window_proportion=0.5'. You *can change this value (between 0..1) to change the size ratio. After changing, you have to restart the application.*

Each component in the Edit-tab has its own function (and properties) and most of the components can be configured to perform a specialised task. A group of components can be selected by pressing the left mousekey, dragging the mouse cursor, and releasing the mousekey again. The mouse cursor shape changes (✥). The rectangle can be moved, again by pressing the left mousekey and dragging. Note, that the mouse cursor must be in the white canvas and not on one of the components. Components can also be deleted as a group, by selecting them as a group and pressing the DELETE key.

ⓘ

By mean of the scrollwheel, the Edit-tab window can be scrolled up and down. By means of the PAGE_UP / PAGE_DOWN the component size can be increased or decreased.

Floating Toolbar



The floating toolbar, is used to add the different types of components to the canvas. The icons on the left represent the types of components that can be added. The right side icons are used for connecting and disconnecting components and context sensitive help.

System component

A System component is the most top level component to which all other components are connected. A System component can be seen as the container that includes the components that are needed to create, display, move and affect particles. The floating toolbar does not contain a System component; it is always present and can also not be deleted from the canvas.

Technique component

With the use of a Technique component it is possible to display particles with different materials (textures) in one System. Each Technique has its own Renderer and uses its own material. By combining multiple Techniques into one System, it is possible to create complex particle systems.

Renderer component

A Renderer is responsible for rendering visual particles. The Renderer determines how the particle will look like. This means that changing a Renderer implies that a totally different particle type is rendered.

Emitter component

Emitters are responsible for spawning (emitting) the particles.

Affector component

Affectors modify (affect) particles over their lifetime.

Observer component

Observers are used to observe whether a certain condition/event occurs. This condition is often related to the state of a particle (is a particle expired?), but also other situations can be validated, for example 'validate whether x amount of time has been exceeded since the particle system was started'.

Event Handler component

*Event Handler*s are used to perform a task in case a certain event happens. An *Event Handler* is associated with an Observer; The Observer signals which event occurs, while the *Event Handler* performs the action.

Behaviour component

A Behaviour component makes it possible to add additional behaviour to a particle. A Slave Behaviour for example, makes it possible to stick particles from one Technique with particles from another Technique (the particle is a slave of the other particle).

Extern component

An Extern component forms a bridge between an external component and a Technique. An example is the PhysX Extern component.

Default cursor

If either the connect or disconnect icon is selected (see below), the cursor changes and has a specific function. If the default cursor icon is selected, the cursor actions are set to default again (selecting).

Connect

After selecting this icon, it is possible to connect two components in the Edit-tab with each other. Click on the first component adds a wire to the component. Selecting another component and the wire is also attached to the other component. Only certain combinations of components are allowed to be connected. If you try to connect components that cannot be connected, a red cross (✖) is displayed in that component.

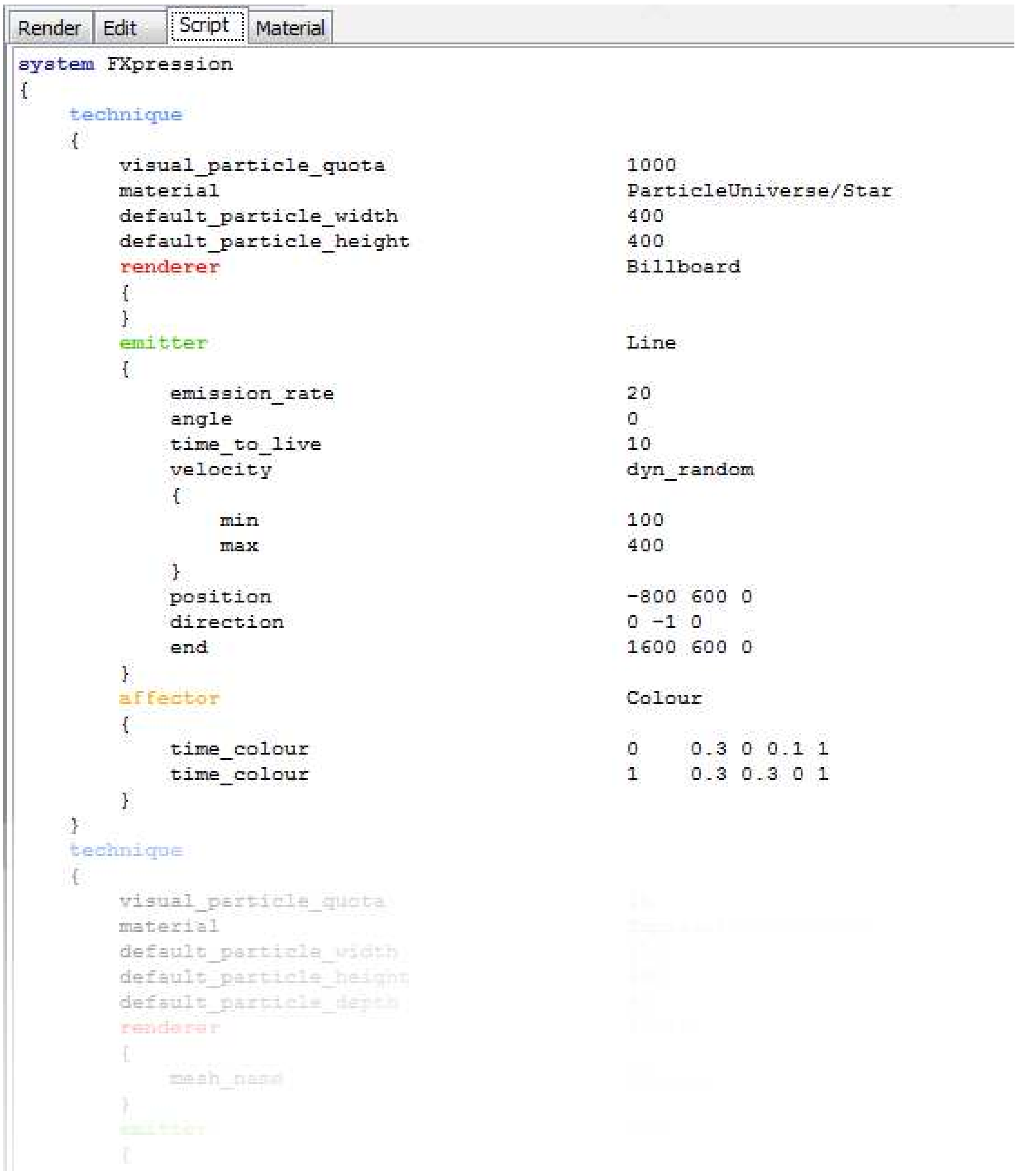
Disconnect

Makes it possible to remove a connection again. When this icon is selected and a (connected) component is selected on the Edit-tab, a listbox appears with the established connections for this component. Picking on from the list removes that connection.

Help

After one of the components in the Edit-tab is selected (has focus) and this icon is selected, a browser window is opened with information about that type of component and its properties. This function is context-sensitive.

Script-tab



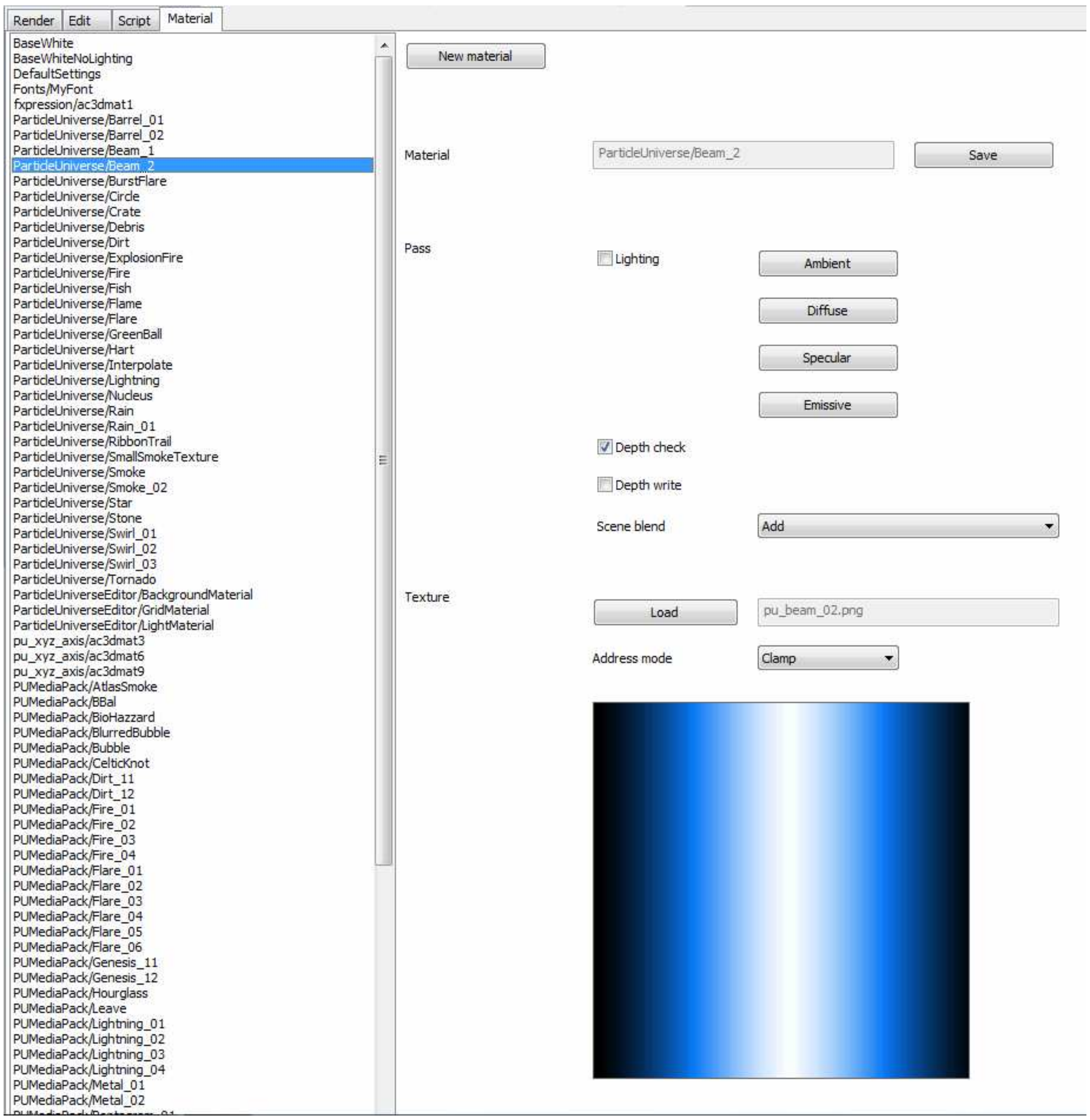
```
system FXpression
{
    technique
    {
        visual_particle_quota                   1000
        material                                ParticleUniverse/Star
        default_particle_width                  400
        default_particle_height                 400
        renderer                                Billboard
        {
        }
        emitter                                 Line
        {
            emission_rate                       20
            angle                               0
            time_to_live                        10
            velocity                            dyn_random
            {
                min                             100
                max                             400
            }
            position                            -800 600 0
            direction                           0 -1 0
            end                                 1600 600 0
        }
        affector                                Colour
        {
            time_colour                         0    0.3 0 0.1 1
            time_colour                         1    0.3 0.3 0 1
        }
    }
    technique
    {
        visual_particle_quota                   [illegible]
        material                                [illegible]
        default_particle_width                  [illegible]
        default_particle_height                 [illegible]
        default_particle_depth                  [illegible]
        renderer                                [illegible]
        {
            mesh_name                           [illegible]
        }
        emitter                                 [illegible]
        {
```

The Script-tab is another representation of the particle system. Particle systems created by means of the Edit-tab can be viewed or changed in the Script-tab. Any change made in the script is synchronised with the Edit-tab representation and vice versa. The scripts can be stored as a *.pu file. Scripts can also be loaded by the plugin within a game. The keywords of the script are highlighted by using the same colours as in the Edit-tab. This option can be turned off in the Options screen.
The Compile icon in the menu bar only has a purpose if this tab is selected.

ⓘ

The script manual – in HTML format – is included in the 'manual'directory and can also be displayed by selecting the ? icon in the floating toolbar of the Edit-tab*.*

Material-tab



BaseWhite
BaseWhiteNoLighting
DefaultSettings
Fonts/MyFont
fxpression/ac3dmat1
ParticleUniverse/Barrel_01
ParticleUniverse/Barrel_02
ParticleUniverse/Beam_1
ParticleUniverse/Beam_2
ParticleUniverse/BurstFlare
ParticleUniverse/Circle
ParticleUniverse/Crate
ParticleUniverse/Debris
ParticleUniverse/Dirt
ParticleUniverse/ExplosionFire
ParticleUniverse/Fire
ParticleUniverse/Fish
ParticleUniverse/Flame
ParticleUniverse/Flare
ParticleUniverse/GreenBall
ParticleUniverse/Hart
ParticleUniverse/Interpolate
ParticleUniverse/Lightning
ParticleUniverse/Nucleus
ParticleUniverse/Rain
ParticleUniverse/Rain_01
ParticleUniverse/RibbonTrail
ParticleUniverse/SmallSmokeTexture
ParticleUniverse/Smoke
ParticleUniverse/Smoke_02
ParticleUniverse/Star
ParticleUniverse/Stone
ParticleUniverse/Swirl_01
ParticleUniverse/Swirl_02
ParticleUniverse/Swirl_03
ParticleUniverse/Tornado
ParticleUniverseEditor/BackgroundMaterial
ParticleUniverseEditor/GridMaterial
ParticleUniverseEditor/LightMaterial
pu_xyz_axis/ac3dmat3
pu_xyz_axis/ac3dmat6
pu_xyz_axis/ac3dmat9
PUMediaPack/AtlasSmoke
PUMediaPack/BBal
PUMediaPack/BioHazzard
PUMediaPack/BlurredBubble
PUMediaPack/Bubble
PUMediaPack/CelticKnot
PUMediaPack/Dirt_11
PUMediaPack/Dirt_12
PUMediaPack/Fire_01
PUMediaPack/Fire_02
PUMediaPack/Fire_03
PUMediaPack/Fire_04
PUMediaPack/Flare_01
PUMediaPack/Flare_02
PUMediaPack/Flare_03
PUMediaPack/Flare_04
PUMediaPack/Flare_05
PUMediaPack/Flare_06
PUMediaPack/Genesis_11
PUMediaPack/Genesis_12
PUMediaPack/Hourglass
PUMediaPack/Leave
PUMediaPack/Lightning_01
PUMediaPack/Lightning_02
PUMediaPack/Lightning_03
PUMediaPack/Lightning_04
PUMediaPack/Metal_01
PUMediaPack/Metal_02

Material ParticleUniverse/Beam_2

Pass
☐ Lighting
☑ Depth check
☐ Depth write
Scene blend Add

Texture pu_beam_02.png
Address mode Clamp

In version 1.4 of the Particle Editor, a new tab was added, the Material-tab. The image/texture used with a particle is ´wrapped´ by a ´material´. The texture is represented by an image file. Several image formats can be used (.png, .dds, jpg, etc.). The material defines how the texture is rendered into a scene. Since Particle Universe is based on the Ogre render engine, it supports all features of the Ogre material system. However, the editor only allows to create single-technique, single-pass and single texture materials. More complicated materials must be created by means of a text editor or some other material editor. In most cases, the simple materials created by the editor are good enough as a basis for particles.

Materials are displayed in a listbox. After selecting a material, its properties are displayed. It also displays the texture that is associated with the selected material.

New material

Create a new ´empty´ material, with default settings and without a texture.

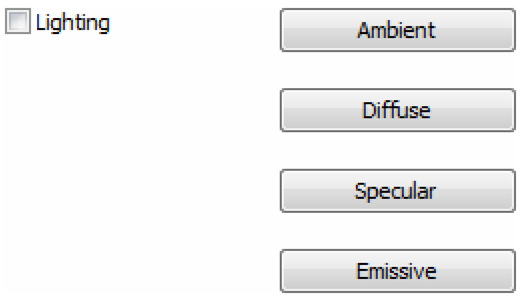
Save material

A material can be save as a file with extension *.material. Ogre allows multiple materials in one file, but the editor can only save a *.material file with 1 material. After the file is saved, the editor takes care of the fact that after the next startup, the material is loaded in the editor. To do this, it adds a resource location, which can be removed again by means of the options screen.

During deployment, all materials should be combined into one or only a few files, to limit the amount of I/O. Note, that the same applies to the particle (.pu) scripts.*

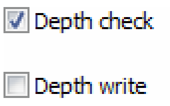
The name of a material can be changed, but this is only possible for materials that are not used by one of the particle systems. If the material is used, the name text box is disabled from input.

Lighting



Sets whether or not dynamic lighting is turned on or not. If ´lighting´ is turned off, the rendered particles will be fully lit. If ´lighting´ is on, the Ambient, Diffuse, Specular and Emissive colours apply.

Depth



Depth check determines whether or not particles are renders with depth-buffer checking on or not. If depth-buffer checking is on, whenever a pixel is about to be

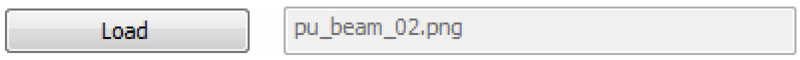
written to the frame buffer the depth buffer is checked to see if the pixel is in front of all other pixels written at that point. If not, the pixel is not written
If the Depth write is set on, whenever a pixel is written to the frame buffer, the depth buffer is updated with the depth value of that new pixel, thus affecting future rendering operations if future pixels are behind this one

Scene blend

Sets the kind of blending particles have with the existing contents of the scene. A few options can be selected:

- **Add**: The colour of the rendering output is added to the scene.
- **Transparent alpha**: The alpha value of the rendering output is used as a mask.
- **Transparent colour**: Colour the scene based on the brightness of the input colours, but don't darken.
- **Modulate**: The colour of the rendering output is multiplied with the scene contents.
- **Replace**:
- **Source colour - One**:
- **Source colour - Zero**:
- **Source colour - Destination colour**:
- **Destination colour - One**:
- **Destination colour - Source colour**:

Load texture



Loads an image file from the file system. After the file has been loaded, it is displayed.

The 'Ogre material' supports more image formats than the Particle Universe editor. Images with dds format cannot be displayed on the Material-tab for example.

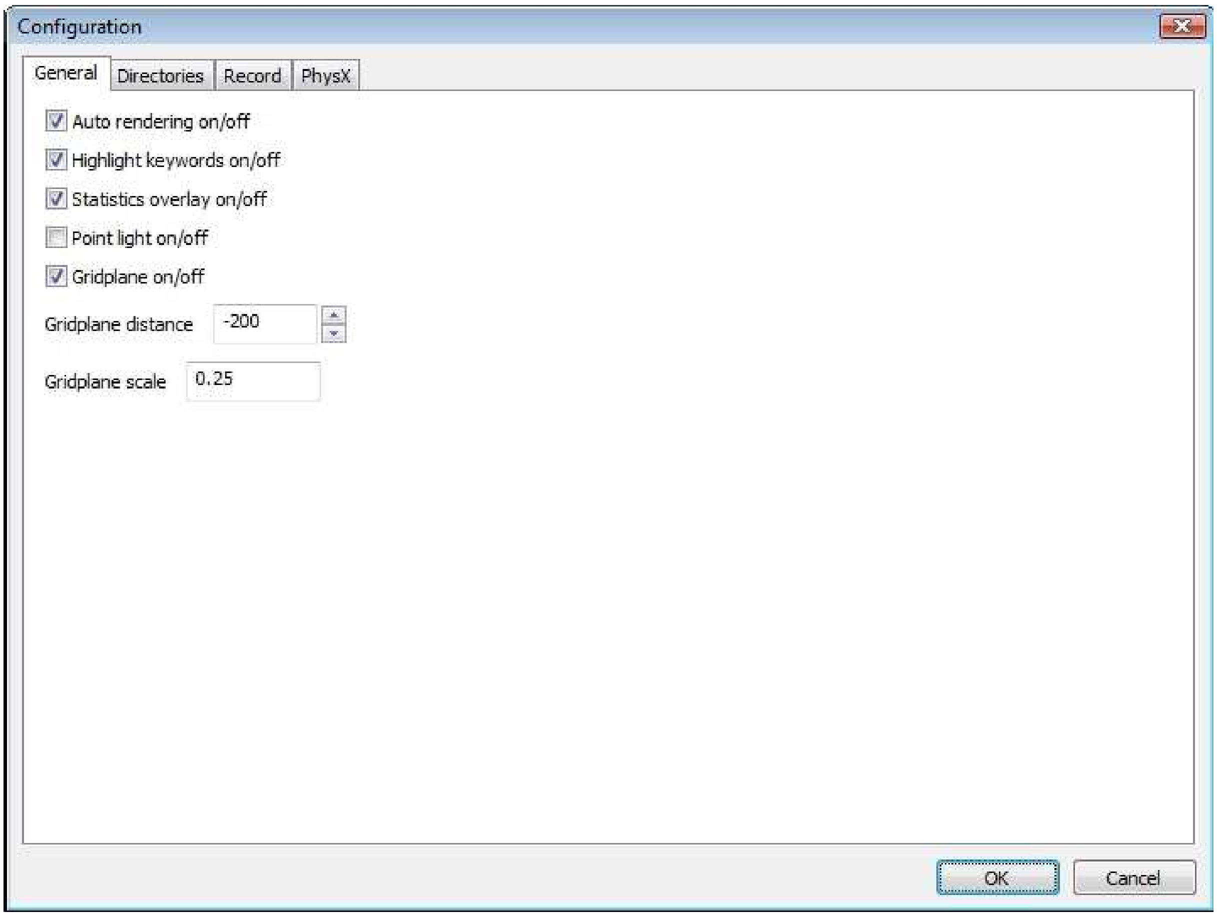
Address mode

Defines what happens when texture coordinates exceed 1.0 for this texture layer. Possible options:

- **Wrap**: Any value beyond 1.0 wraps back to 0.0. The texture is repeated.
- **Mirror**: Texture flips every boundary, meaning texture is mirrored every 1.0 u or v.
- **Clamp**: Values beyond 1.0 are clamped to 1.0. Texture 'streaks' beyond 1.0 since last line of pixels is used across the rest of the address space.

Options

The ‘Options’ window can be found in the menu, in ‘Tools’ > ‘Options’ and contains settings that are used every time the application is started. It consists of four tabs.

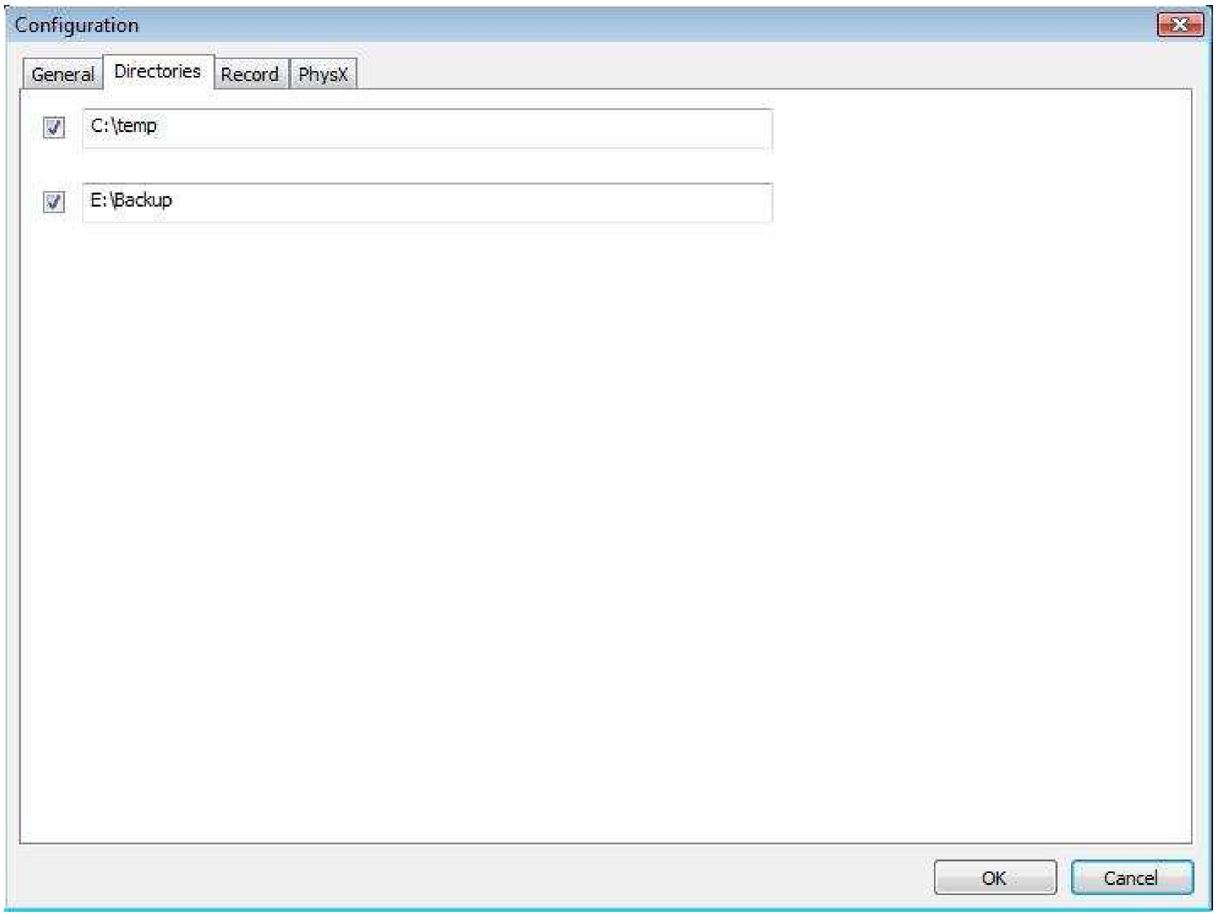


- [✓] Auto rendering on/off
- [✓] Highlight keywords on/off
- [✓] Statistics overlay on/off
- [ ] Point light on/off
- [✓] Gridplane on/off

Gridplane distance -200

Gridplane scale 0.25

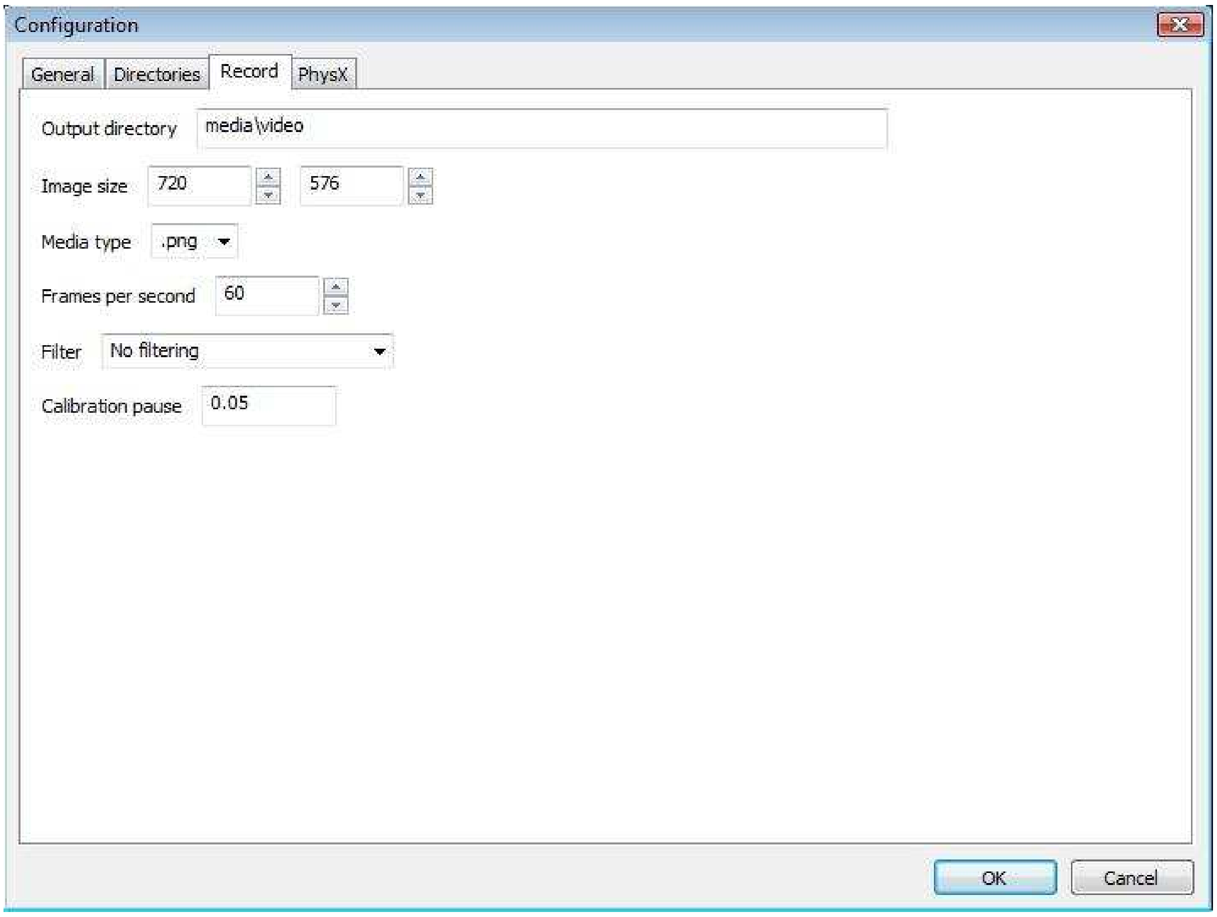
- **Auto rendering on/off** – If the Render-tab on the main screen is selected, the particle system starts automatically is this option is checked.
- **Highlight keywords on/off** – By default, the keywords in the Script-tab are highlighted with a certain colour. With this option, the highlight feature can be turned on or off.
- **Statistics overlay on/off** – If checked, the statistics are displayed on the screen. When the editor is started, it uses this value to display or hide the statistics.
- **Point light on/off** – By default, if the editor is started, no point light is added to the scene.
- **Gridplane on/off** – By default, if the editor is started, a gridplane is added to the scene.
- **Gridplane distance** – Setting the height (y-value) of the plane.
- **Gridplane scale** – If the scale is set higher, the grid contains larger tiles.



[✓] C:\temp

[✓] E:\Backup

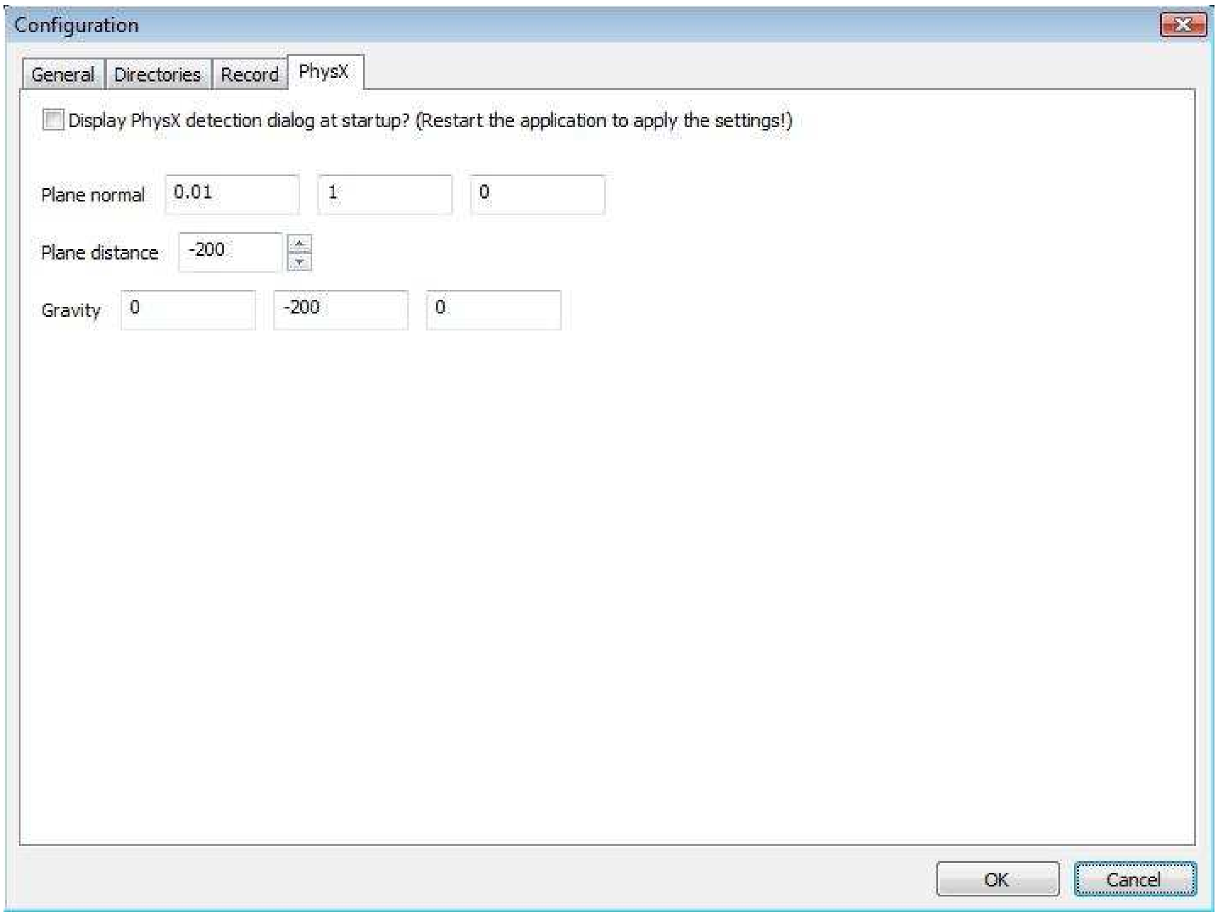
If new resource locations are added, they are listed in this tab. Unchecking them means that these resource locations are not used anymore, so scripts, meshes, materials and textures contained in these directories, are not visible anymore.



Output directory media\video
Image size 720 576
Media type .png
Frames per second 60
Filter No filtering
Calibration pause 0.05

The 'Record' tab contains settings used for recording (exporting) a particle system to a series of image files (.png, .jpg, .tiff) or video file (.avi,.swf (Small Web Format), .wmv (Windows Media Video)).

- **Output directory** – Directory that contains the generated files.
- **Image size** – The width and height of the image files.
- **Media type** – .png, .jpg, .tiff, .avi, .swf or .wmv files can be generated. If .avi, .swf or wmv is selected, a new selection box is presented from which the size can be selected; only a number of prefixed sizes are possible. The 'Image size' is disabled in that case.
- **Frames per second** – determines the speed of the animation. It generates more image files if this setting is higher.
- **Filter** – 3 types are supported:
    - *No filtering*
    - *Alpha from luminance*; an alpha value is added. The alpha value is calculated based on the intensity of the RGB values. Use the .png or .tiff format if the images are generated with an alpha channel.
    - *Alpha from background colour*; The alpha value is set to maximum where the pixel colour of the rendered image is equal to the background colour. Only applicable for .png and .tiff formats.
- **Calibration pause** – Recording is not exactly performed in runtime, because it takes some time to generate the image file, especially when the image size is large. This setting takes a brief pause after writing each image file, to ensure that there is no jittering in the image sequence.


Configuration

General | Directories | Record | PhysX

☐ Display PhysX detection dialog at startup? (Restart the application to apply the settings!)

Plane normal 0.01 1 0

Plane distance -200

Gravity 0 -200 0

OK Cancel

The PhysX tab includes settings for usage of Nvidia's PhysX library. The use of PhysX is optional. These configuration options are only displayed when PhysX has been selected during the install.

- **Display PhysX detection dialog** – When the Particle Universe Editor is started for the first time, it detects whether PhysX is installed or not. It gives the user the opportunity to enable or disable the PhysX features (this setting is used each time the application is started). The dialog is only displayed once, but can be activated again by means of this checkbox. A restart of the Particle Universe Editor is required.
- **Plane Normal** – If a particle system with PhysX attributes is started, a plane is used with which the PhysX-enabled particles collide. By defaults, this is a horizontal plane, but the plane's normal can be changed by means of this tab.
- **Plane distance** – Defines the distance from the centre of the particle system to the plane.
- **Gravity** – Defines the gravity of the scene in which the particle systems lives.

Multilingual

From Particle Universe version 1.5.1 the editor supports multiple languages. This comes to effect during installation of the Particle Universe (the setup). However, if the application is installed, it is still possible to change the language without reinstalling. Just edit the 'language' property in the pued.cfg file. Assign the language you want to use according to the table below:

Language	Language =
Chinese (simplified)	zh_CN
Dutch	nl_NL
Macedonian	mk_MK
Russian	ru_RU

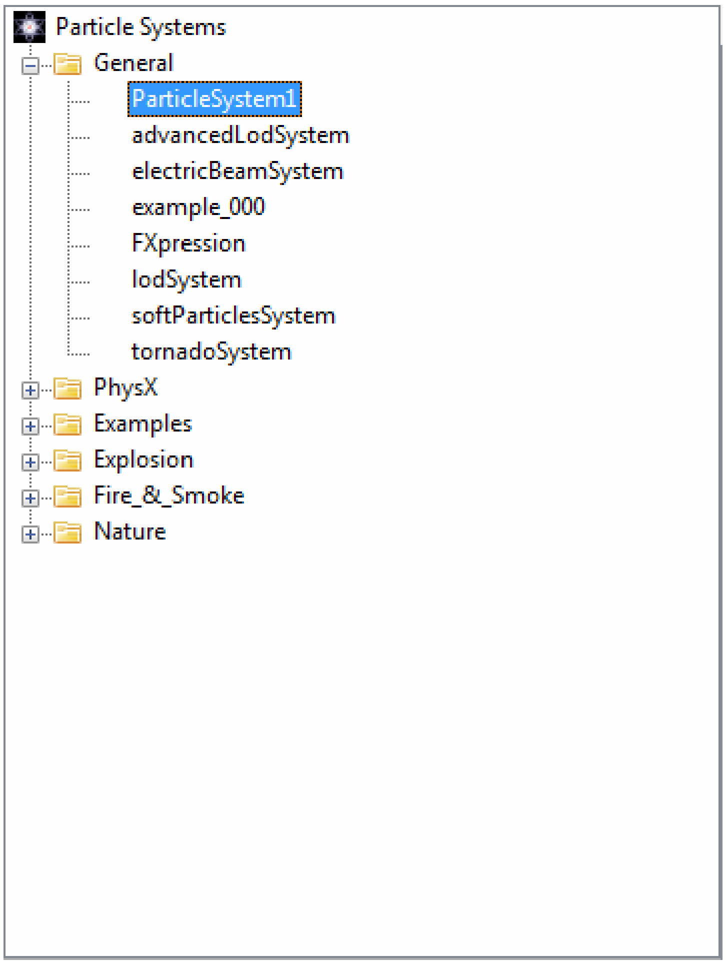
Note, that changing the language cannot be done by means of 'Options' in the application.

ⓘ
*The number of supported languages is still limited. If you want to make your own translation, this is also possible. Just take one of the .po files in the /languages/nl, /languages/mk or any other subdirectory. Use an editor such as poedit (download from www.poedit.net), create a translation and save the *_internat.mo file to the /languages directory (where * is the country code). Edit the pued.cfg file and assign the appropriate language.*

Lesson 1 - A basic particle system

In this first lesson, a basic particle system is created. The actions are presented in small steps. In subsequent chapters the steps are larger.

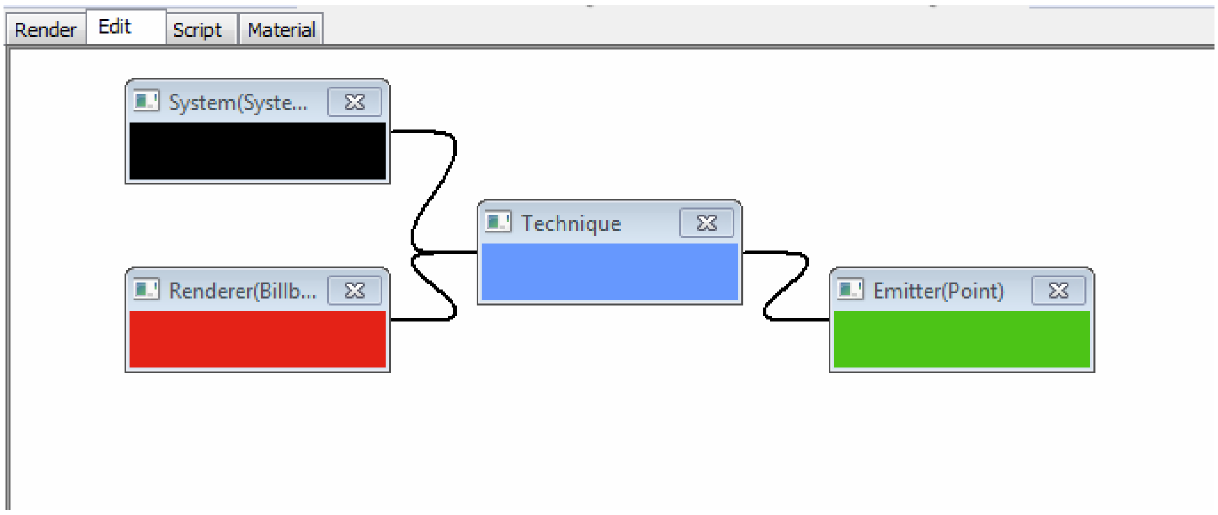
First, create a new particle system by pressing the ‘New’ icon on the left.



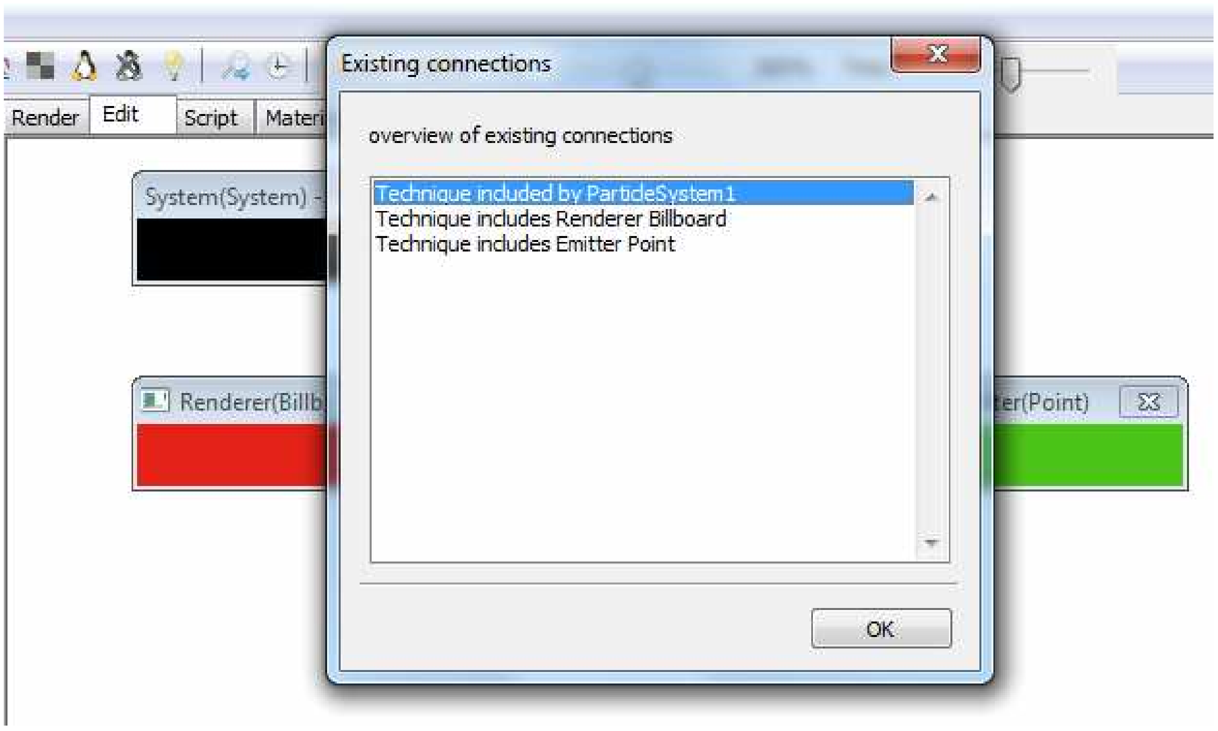
The basic components of which particle systems exist, are the System component itself (of course), a Technique-, an Emitter, and a Renderer component. Any new particle system, created with the editor, already consist of these components.

The Edit-tab contains a basic particle system with the four components..

The System component is always present and cannot be removed.



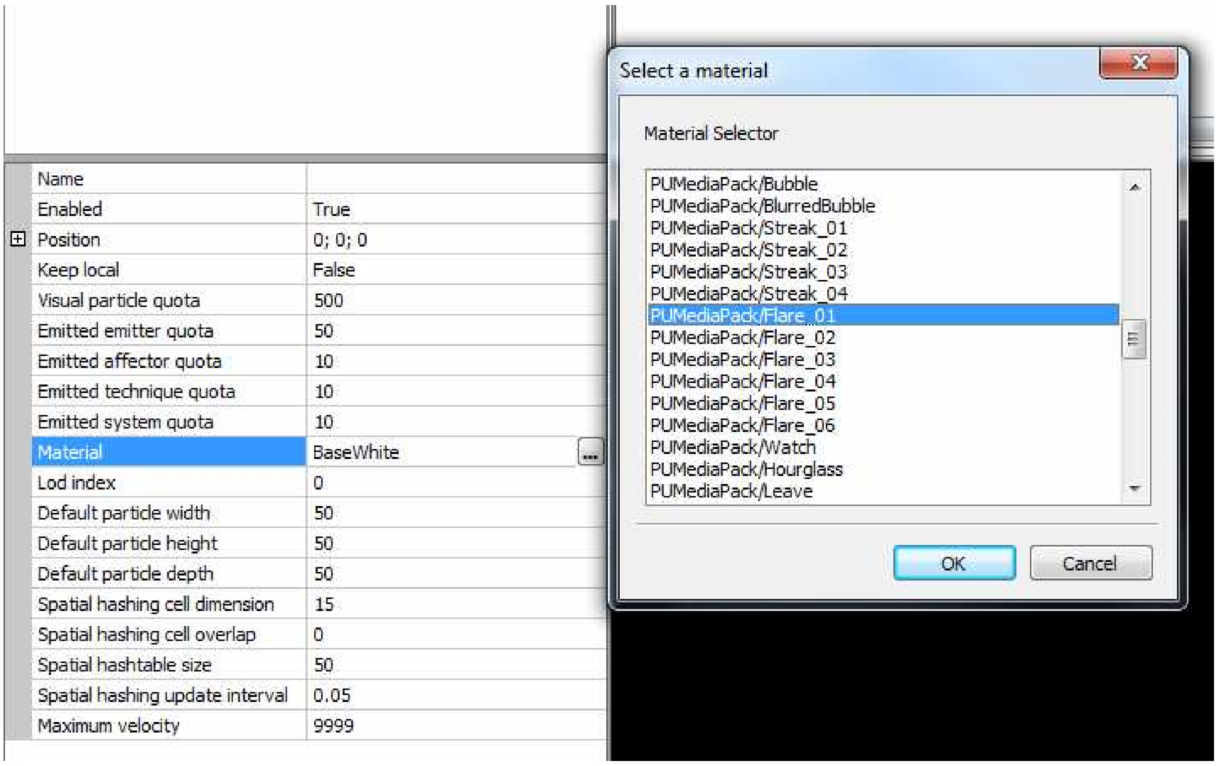
The components are connected with each other. Right-click with the mouse on the Technique component. A dialog is displayed with the type of connections the Technique component has with other components.



overview of existing connections

Technique included by ParticleSystem1
Technique includes Renderer Billboard
Technique includes Emitter Point

Press the OK button and the dialog closes. Left–click on the Technique and notice that the properties of the Technique are displayed on the left part of the screen. Change the ***Material*** property. Select *PUMediaPack/Flare_01*.



Name	
Enabled	True
⊞ Position	0; 0; 0
Keep local	False
Visual particle quota	500
Emitted emitter quota	50
Emitted affector quota	10
Emitted technique quota	10
Emitted system quota	10
Material	BaseWhite
Lod index	0
Default particle width	50
Default particle height	50
Default particle depth	50
Spatial hashing cell dimension	15
Spatial hashing cell overlap	0
Spatial hashtable size	50
Spatial hashing update interval	0.05
Maximum velocity	9999

Select a material

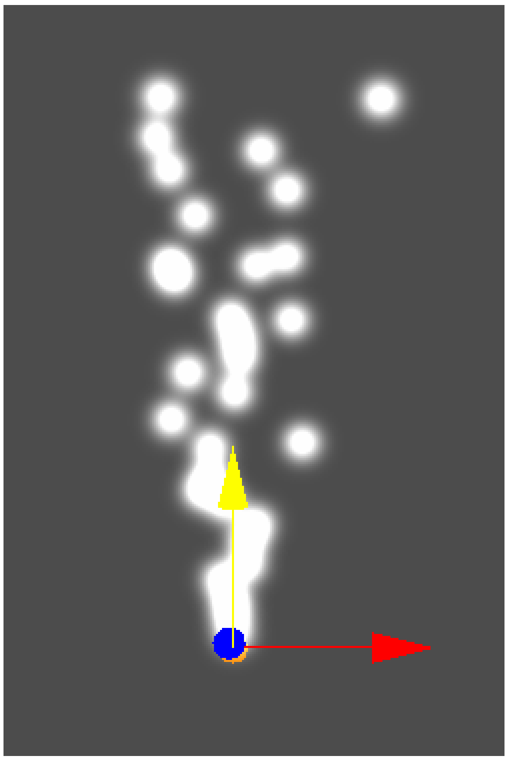
Material Selector

PUMediaPack/Bubble
PUMediaPack/BlurredBubble
PUMediaPack/Streak_01
PUMediaPack/Streak_02
PUMediaPack/Streak_03
PUMediaPack/Streak_04
PUMediaPack/Flare_01
PUMediaPack/Flare_02
PUMediaPack/Flare_03
PUMediaPack/Flare_04
PUMediaPack/Flare_05
PUMediaPack/Flare_06
PUMediaPack/Watch
PUMediaPack/Hourglass
PUMediaPack/Leave

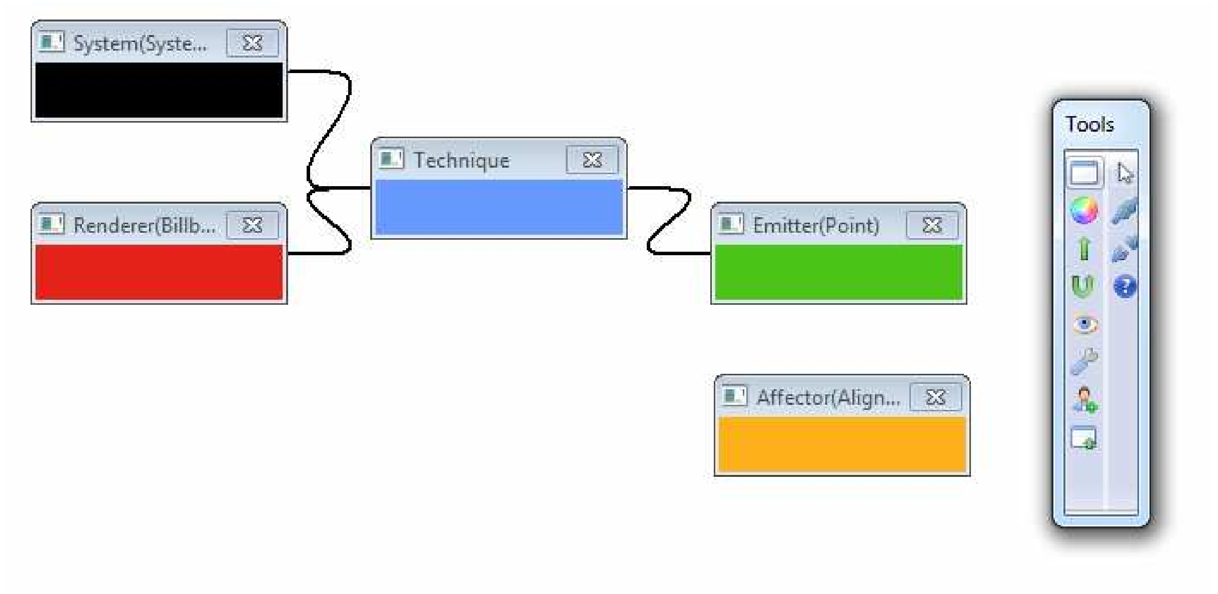
Look at the script in the Script-tab.

```
system ParticleSystem1
{
    Technique
    {
        material                                PUMediaPack/Flare_01
        renderer                                Billboard
        {
        }
        emitter                                 Point
        {
        }
    }
}
```

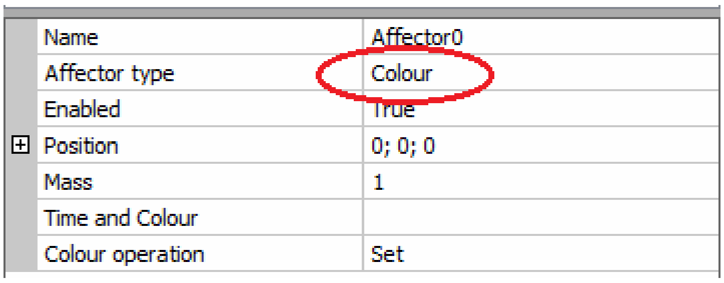
Go back to the Edit-tab or the Render-tab and start the particle system by pressing the icon. It should display something like this.



Go to the Edit-tab and add a new Affector by clicking on the icon on the toolbar.

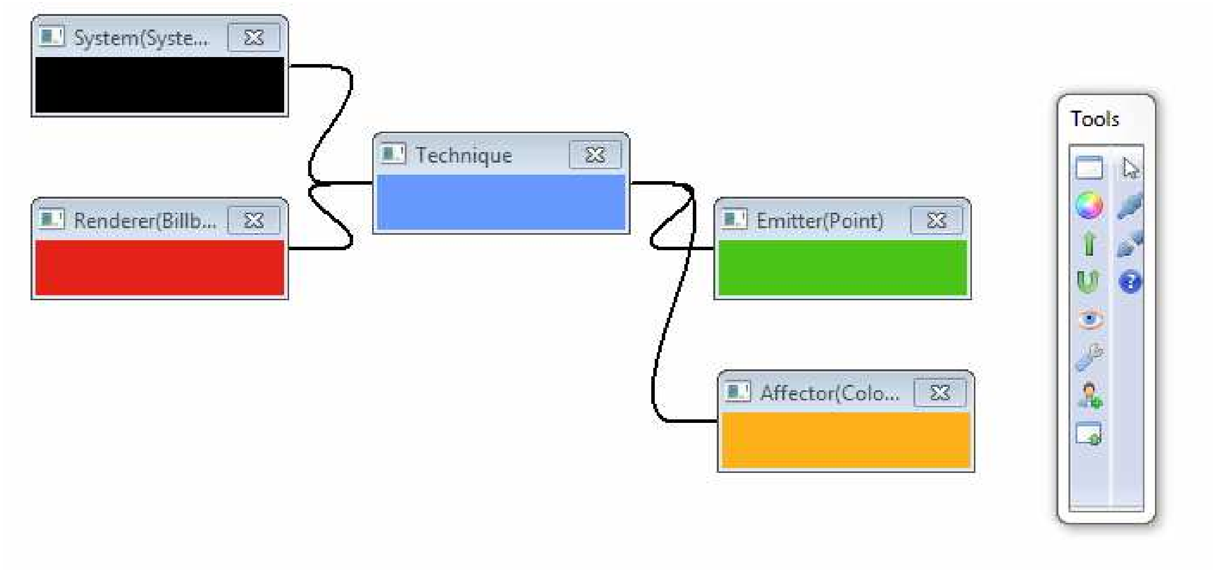


Click on the Affector component and change the ***Affector type*** in the property window from *Align* (= default) into *Colour*.

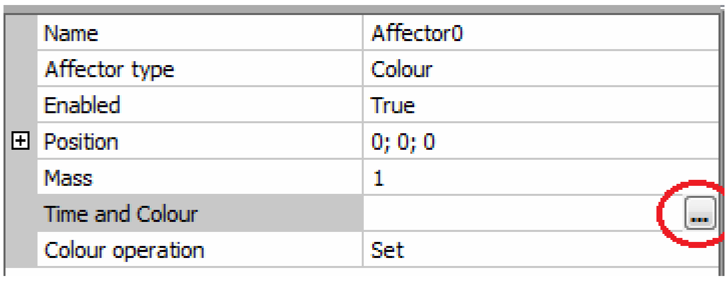


Name	Affector0
Affector type	Colour
Enabled	True
⊞ Position	0; 0; 0
Mass	1
Time and Colour	
Colour operation	Set

The new Colour Affector must be connected to the Technique. Connecting two components is done by pressing the connection icon (). Select the Technique component, move the mouse cursor (notice the wire that is attached to the mouse cursor) and select the Colour Affector component. Both components are now connected.

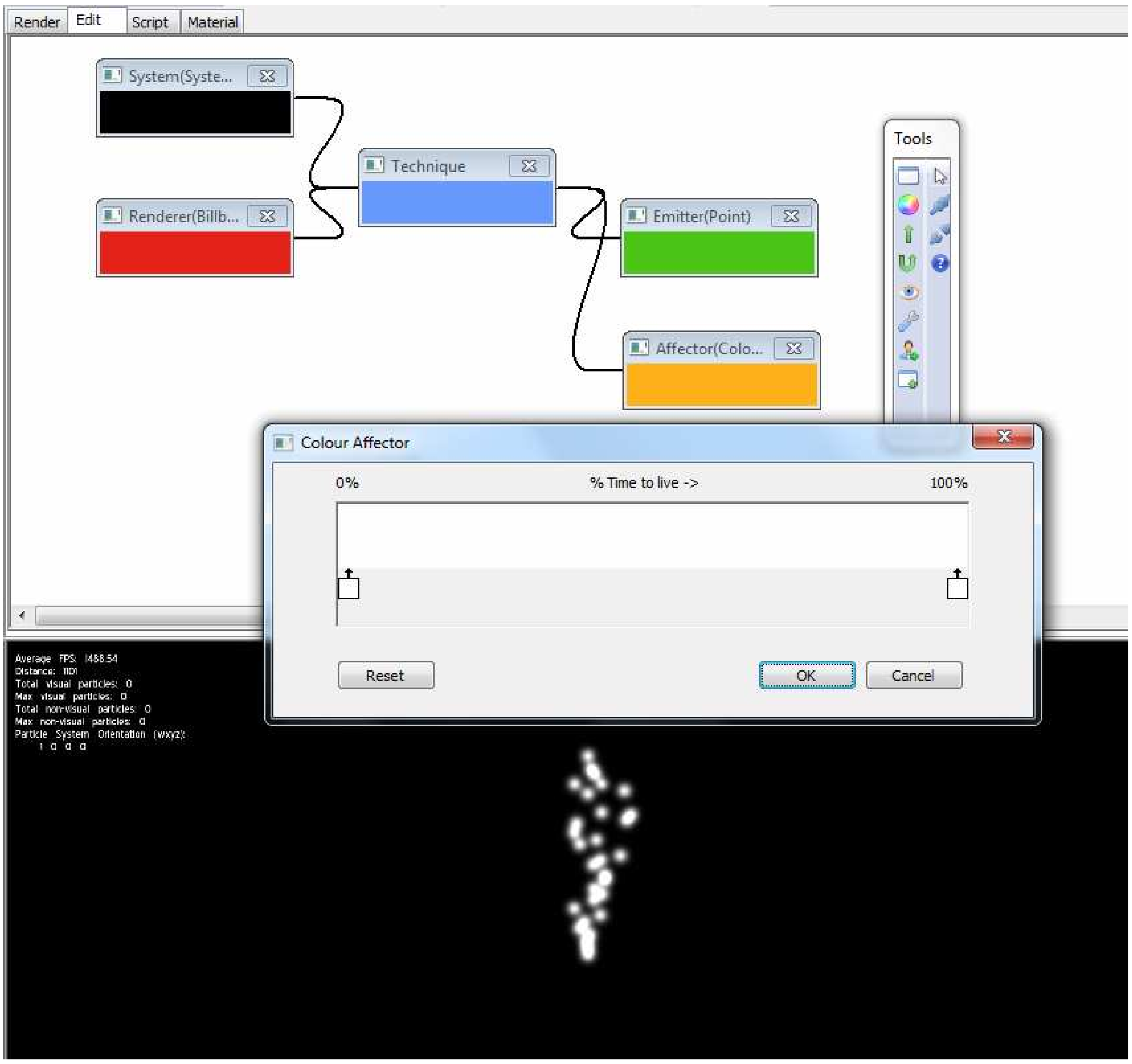


The Colour Affector makes it possible to define colour changes during the lifetime of a particle. Click on the Colour Affector and select ***Time and Colour*** in the property window. A button appears; press it.



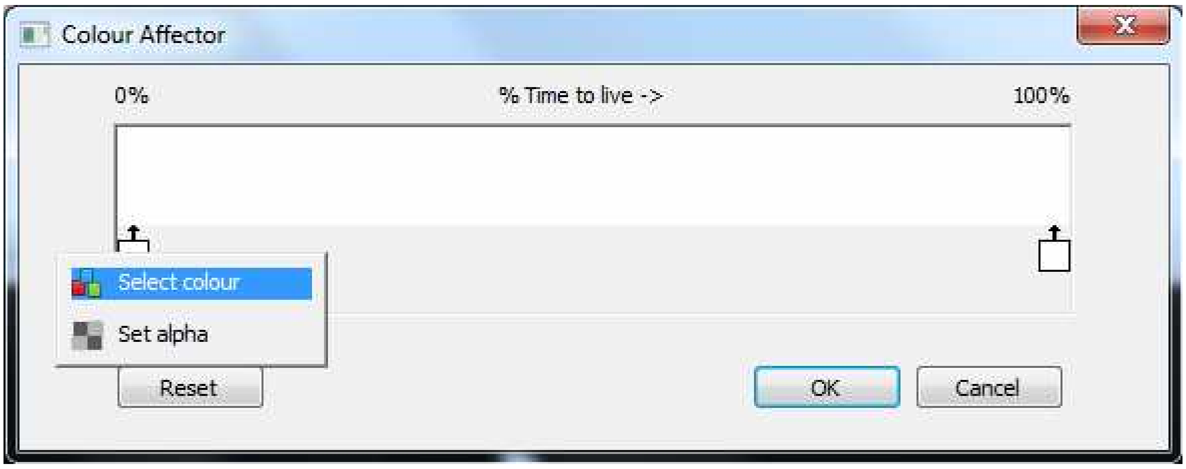
Name	Affector0
Affector type	Colour
Enabled	True
⊞ Position	0; 0; 0
Mass	1
Time and Colour	...
Colour operation	Set

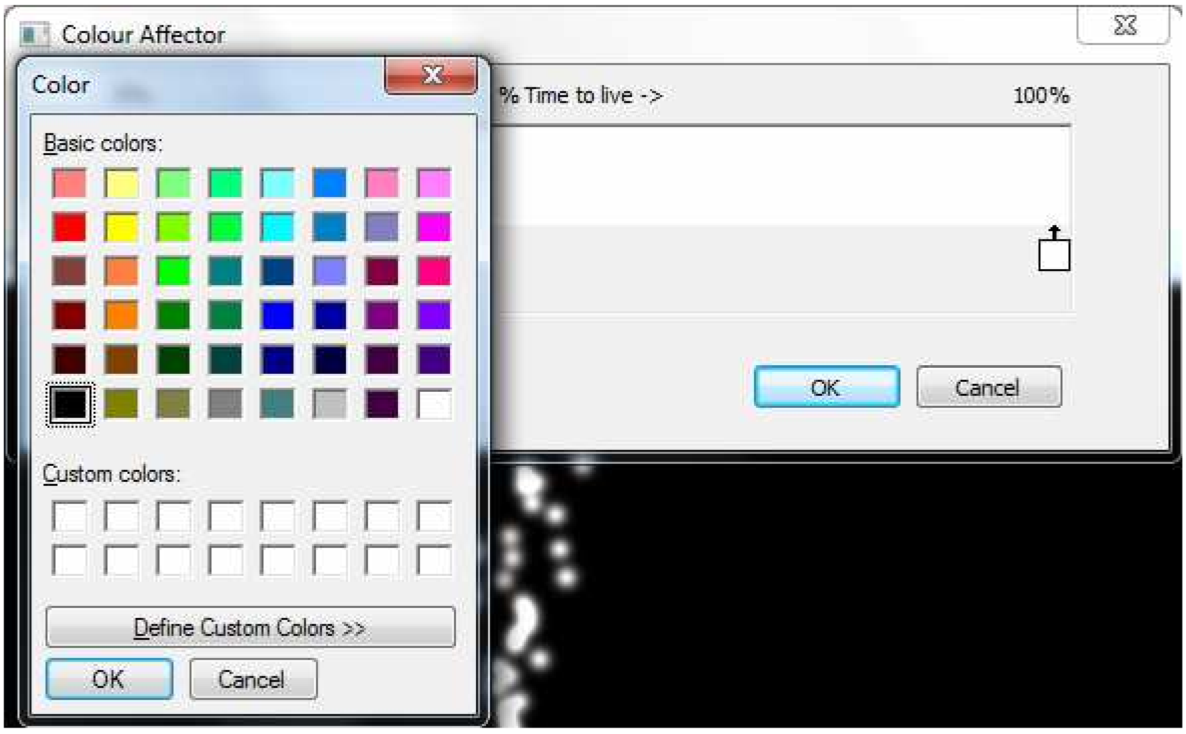
The Colour Affector dialog is displayed.



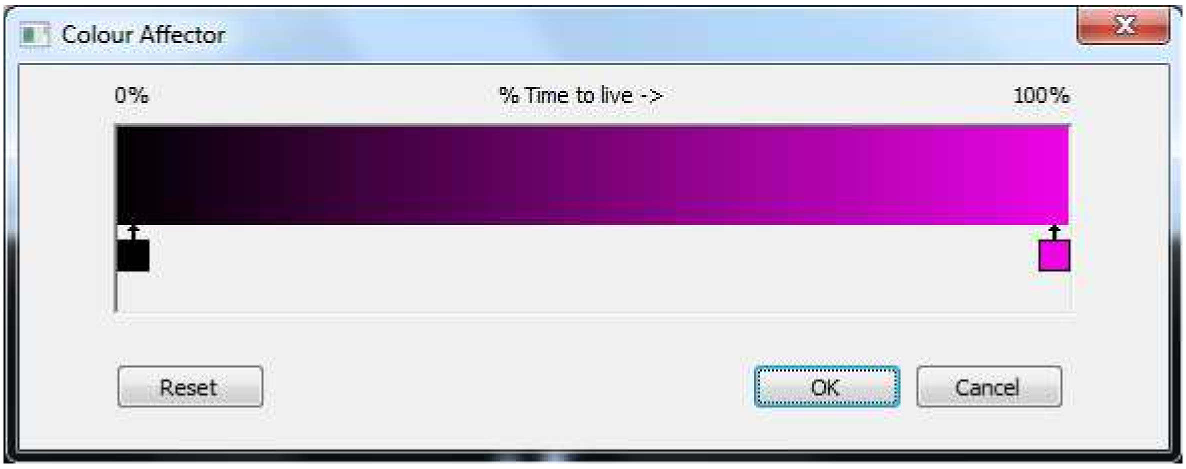
```
Average FPS: 1488.54
Distance: 1101
Total visual particles: 0
Max visual particles: 0
Total non-visual particles: 0
Max non-visual particles: 0
Particle System Orientation (wxyz):
     1 0 0 0
```

You can add as much colourpoints to the dialog as you want (don't exaggerate) and give each one a colour and alpha value. For now, we use the two default points at the left and the right end. Right-click with the mouse on the left colourpoint. Choose ***Select colour***. A new colourpicker dialog is displayed. Select the black colour.

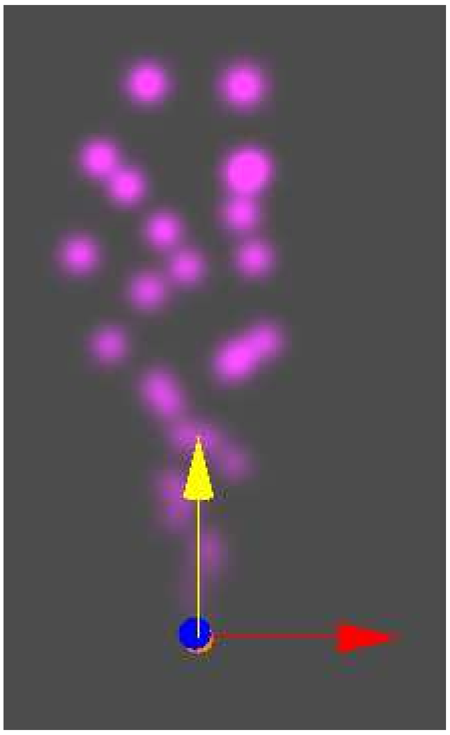




Do the same for the right colourpoint and choose a purple colour.

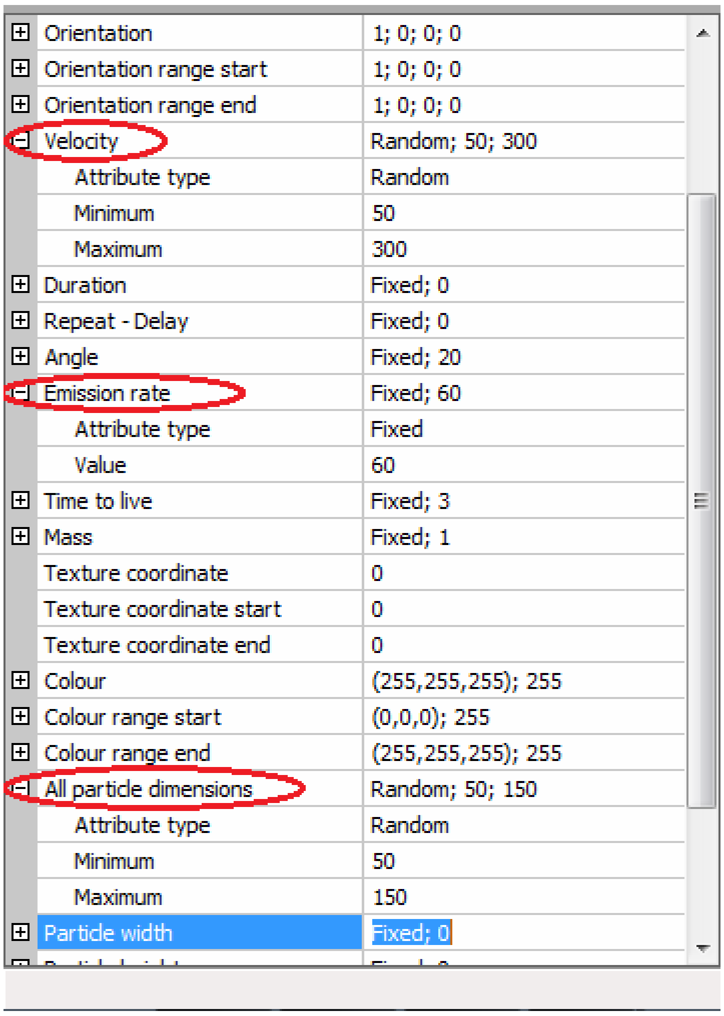


Accept the changes by means of OK and ´play´ the particle system (if not already playing). Something similar as the figure below should be displayed.



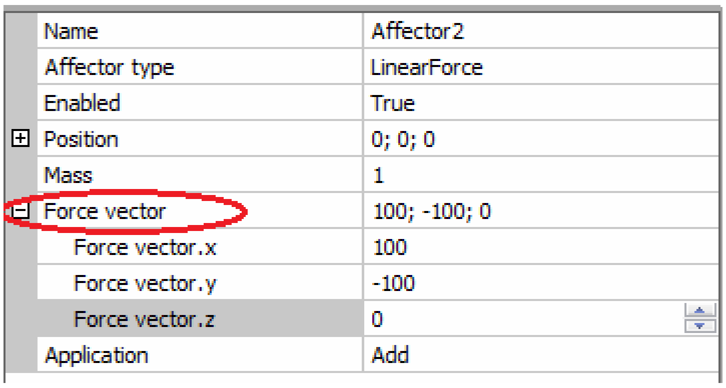
Now, let’s experiment, and change some more properties. Select the Emitter and change the properties ***Emitter type***, ***Velocity***, ***Emission rate*** and ***All particle dimensions***. The last three properties are so-called ‘dynamic attributes’. This means that its value can be:

- a fixed value.
- a random value between min and a max.
- a value that follows a curve; the curve is determined by means of several control points, which are interpolated (Linear or Spline).
- an oscillating value; the value is generated by means of a sine or square function.



⊞ Orientation	1; 0; 0; 0
⊞ Orientation range start	1; 0; 0; 0
⊞ Orientation range end	1; 0; 0; 0
⊟ Velocity	Random; 50; 300
Attribute type	Random
Minimum	50
Maximum	300
⊞ Duration	Fixed; 0
⊞ Repeat - Delay	Fixed; 0
⊞ Angle	Fixed; 20
⊟ Emission rate	Fixed; 60
Attribute type	Fixed
Value	60
⊞ Time to live	Fixed; 3
⊞ Mass	Fixed; 1
Texture coordinate	0
Texture coordinate start	0
Texture coordinate end	0
⊞ Colour	(255,255,255); 255
⊞ Colour range start	(0,0,0); 255
⊞ Colour range end	(255,255,255); 255
⊟ All particle dimensions	Random; 50; 150
Attribute type	Random
Minimum	50
Maximum	150
⊞ Particle width	Fixed; 0

Also add a new Affector and connect it to the Technique. Change its type into LinearForce and set the ***Force vector*** to 100; -100; 0.

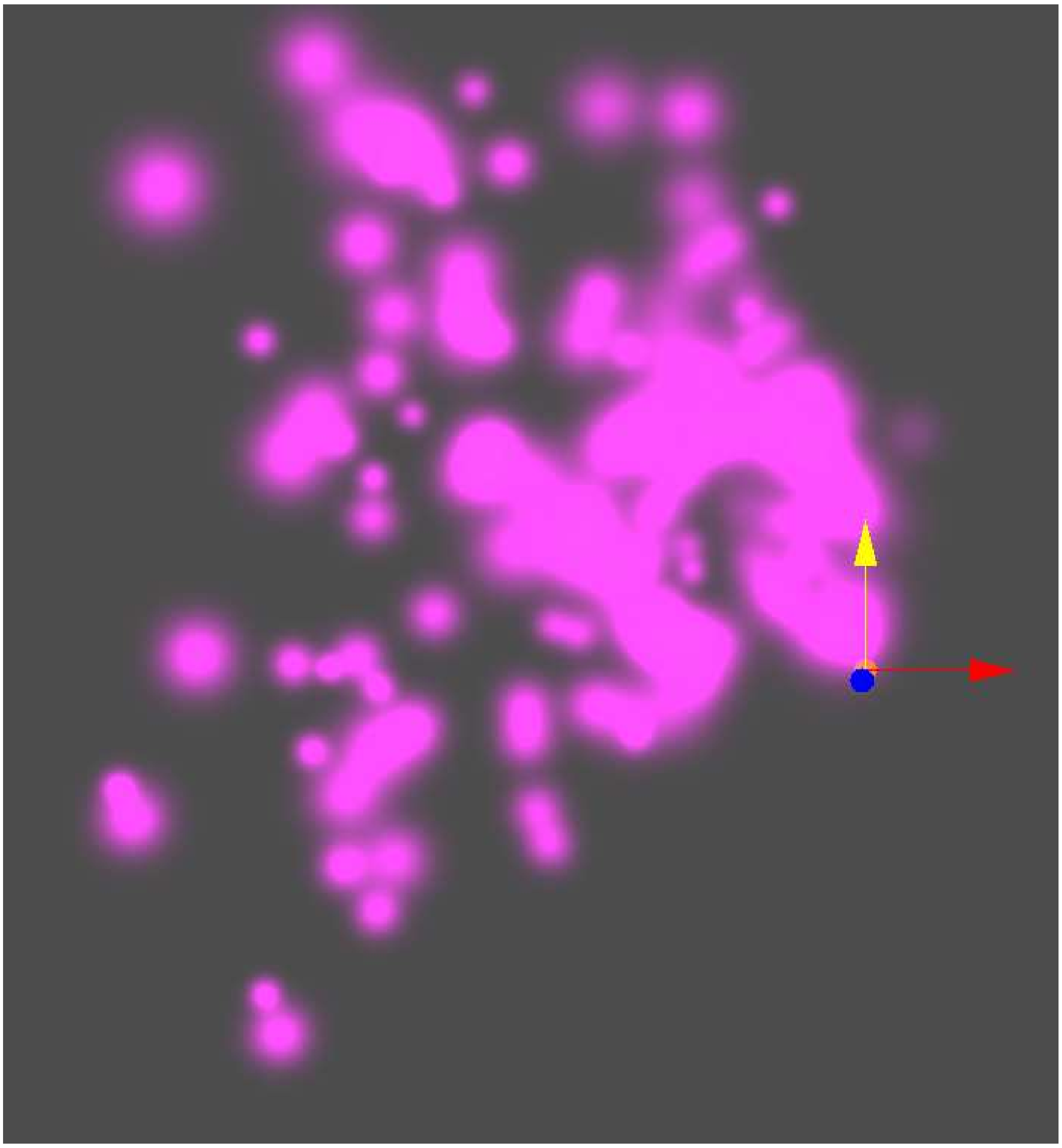


Name	Affector2
Affector type	LinearForce
Enabled	True
⊞ Position	0; 0; 0
Mass	1
⊟ Force vector	100; -100; 0
Force vector.x	100
Force vector.y	-100
Force vector.z	0
Application	Add

The script looks like the one below, still with the pregenerated names for the affectors.

```
system ParticleSystem1
 {
    Technique
    {
        material                                PUMediaPack/Flare_01
        renderer                                Billboard
        {
        }
        emitter                                 Point
        {
            emission_rate                       60
            velocity                            dyn_random
            {
                min                             50
                max                             300
            }
            all_particle_dimensions             dyn_random
            {
                min                             50
                max                             150
            }
        }
        affector                                Colour Affector1
        {
            time_colour                         0    0 0 0 1
            time_colour                         1    0.929412 0.0196078 0.909804 1
        }
        affector                                LinearForce Affector2
        {
            force_vector                        100 -100 0
        }
    }
 }
```

Playing the particle system should display something similar to this.



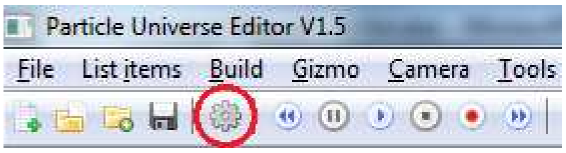
Now return to the Edit-tab.

- Change the ***Name*** property of the Technique into *main_technique*.
- Change the ***Name*** of the Emitter into *emit_flares*.
- Change the ***Name*** of the Colour Affector into *change_colour*
- Change the ***Name*** of the LinearForce Affector into *apply_force*.
- Set the ***Category*** of the System to *Lessons.*

A quick view on the Script-tab displays.

```
system ParticleSystem1
 {
    category                                    Lessons
    technique main_technique
    {
        material                                PUMediaPack/Flare_01
        renderer                                Billboard
        {
        }
        emitter                                 Point emit_flares
        {
            emission_rate                       60
            velocity                            dyn_random
            {
                min                             50
                max                             300
            }
            all_particle_dimensions             dyn_random
            {
                min                             50
                max                             150
            }
        }
        affector                                Colour change_colour
        {
            time_colour                         0    0 0 0 1
            time_colour                         1    0.929412 0.0196078 0.909804 1
        }
        affector                                LinearForce applyForce
        {
            force_vector                        100 -100 0
        }
    }
 }
```

Now change the name of the particle system in the script into *Lesson_1* and compile the script, to check whether there are any errors. Select the compile ( ) icon on the menu bar.



Look at the treelist on the left of the screen and notice that both the name is changed into Lesson_1 and the location of the particle system has been changed. It is moved to the Lessons category.

Save the script by means of the menu option **Save** or use CTRL+S. Choose a certain save location and name the script *Lesson_1.pu*

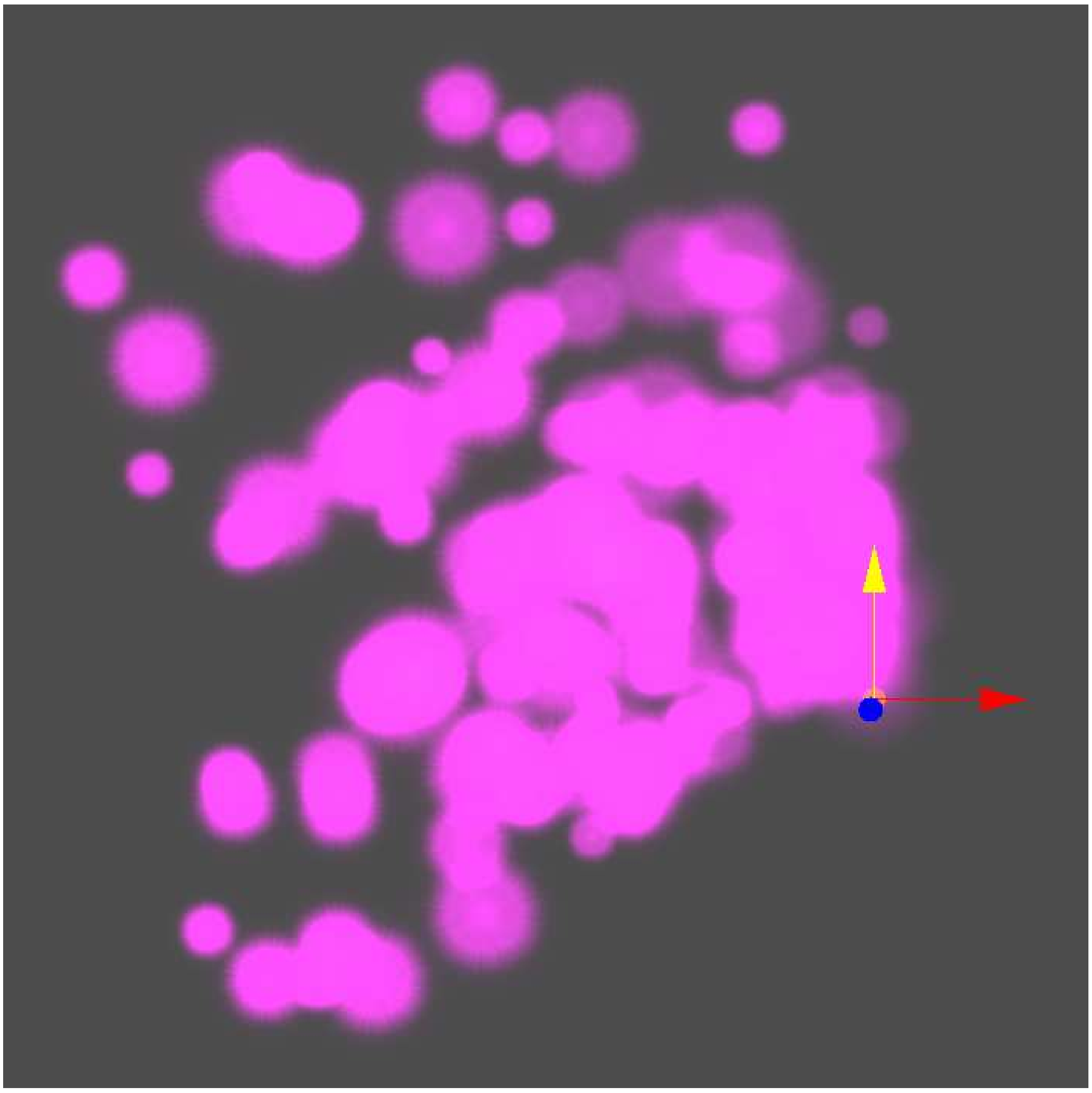
Unlike saved material files (see next chapter), the Lesson_1.pu script is not automatically loaded after a restart of the editor. The Resource location (the directory where the Lesson_1.pu script is saved) must be added manually if needed.

Lesson 2 - Materials

We are reusing the particle system created in the previous lesson. Go to the treelist and select *Lesson_1* in the *Lessons* category and clone the particle system, by means of a right mouseclick and selecting the option **Clone** from the context menu. Rename the particle system into *Lesson_2*. Also use another material, *PUMediaPack/Flare_02*. Use the Edit-tab or Script-tab for this.

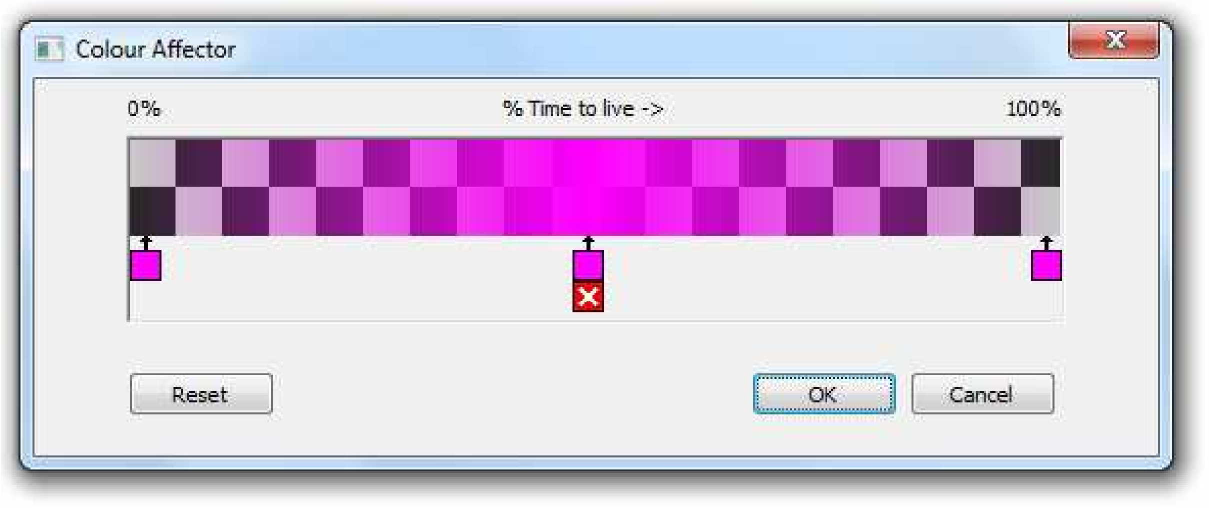
Go to the Material-tab and select *PUMediaPack/Flare_02* from the listbox.
The material name cannot be changed, because it is used in a particle system.
The ***Scene blend*** property has value *Add*. Change it into *Transparent alpha* and play the particle system in the Edit-tab.

It is noticeable that the particles start with a black colour and end with a purple colour. This does not look very good.



Select the Colour Affector and click on the ***Time and Colour*** button in the property window. Set the left colourpoint to the same purple colour and add a new colourpoint in the middle just by clicking (left mouse button) on the *Colour Affector* dialog. Also assign the same purple colour.

Because we want to fade in and out, using the alpha value, right-click on the left colourpoint and set the alpha value to 0. Do the same for the right colourpoint. The dialog looks like this:

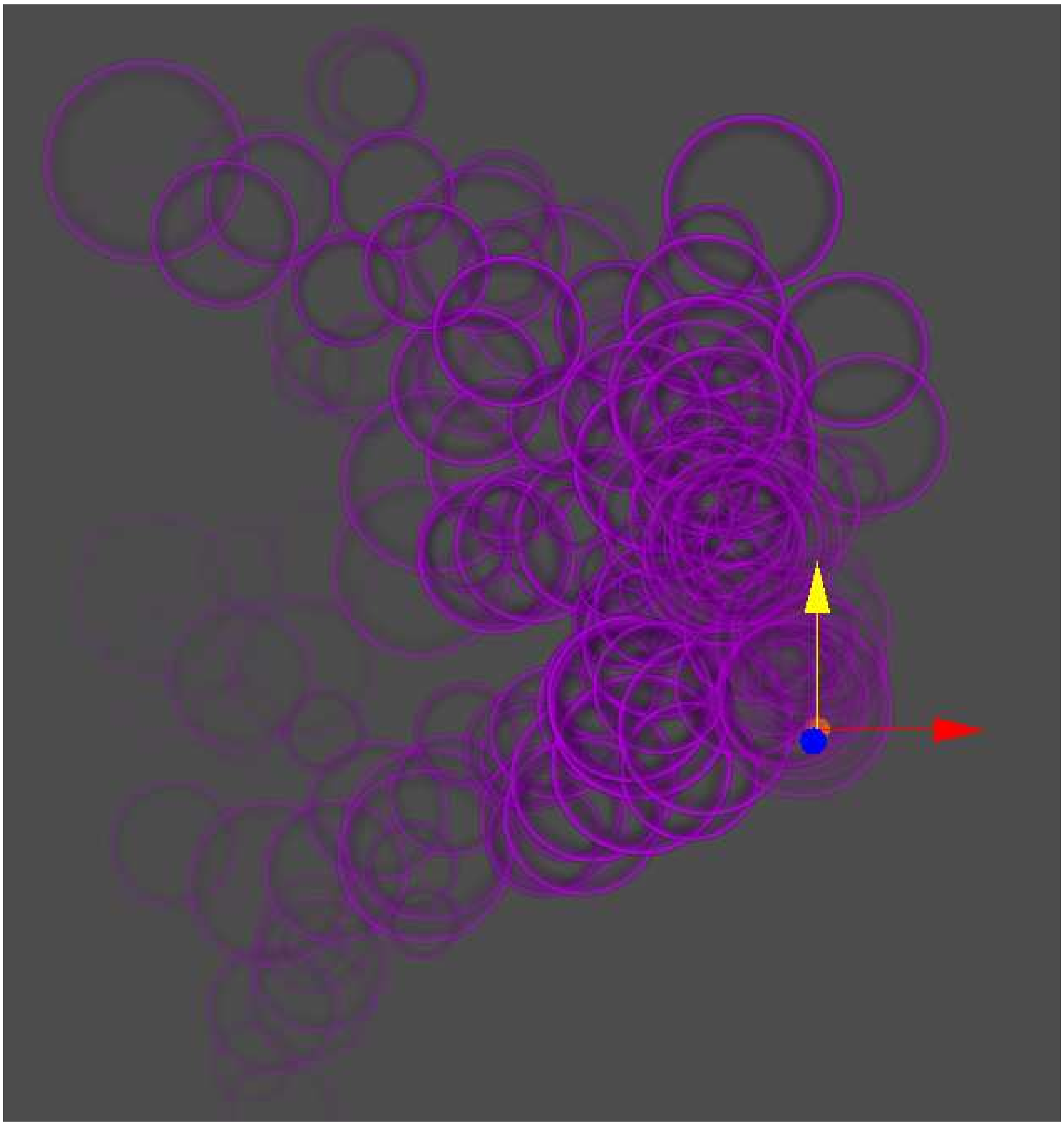


Press OK and see that the particle fades in to the purple colour and slowly fades out again.

Now we want to use a totally new material. Go to the Material-tab and click the **New material** button. It creates a default material with name *Material0*. Rename it to *Lesson_2_Material*. This material does not contain any texture, so we also load a texture. Press the **Load** button and choose

```
…\ParticleUniverseEditor\media\ParticleUniverse\mediapack\textures\pump_ring_03.png
```

Disable ***Lighting***, ***Depth check*** and ***Depth write***. Set ***Scene blend*** to *Transparent alpha*. Change the material in the Technique by changing the ***Material*** to *Lesson_2_Material* and play the particle system.



Save the material. Go to the Material-tab and press the **Save** button. The default name is *Lesson_2_Material.material*

Open the Options dialog via the menu (or CTRL-O). The Directories tab contains both the resource paths of the texture (*pump_ring_03.png*) and the material. The next time the editor is started, *Lesson_2_Material.material* and *pump_ring_03.png* are automatically loaded.

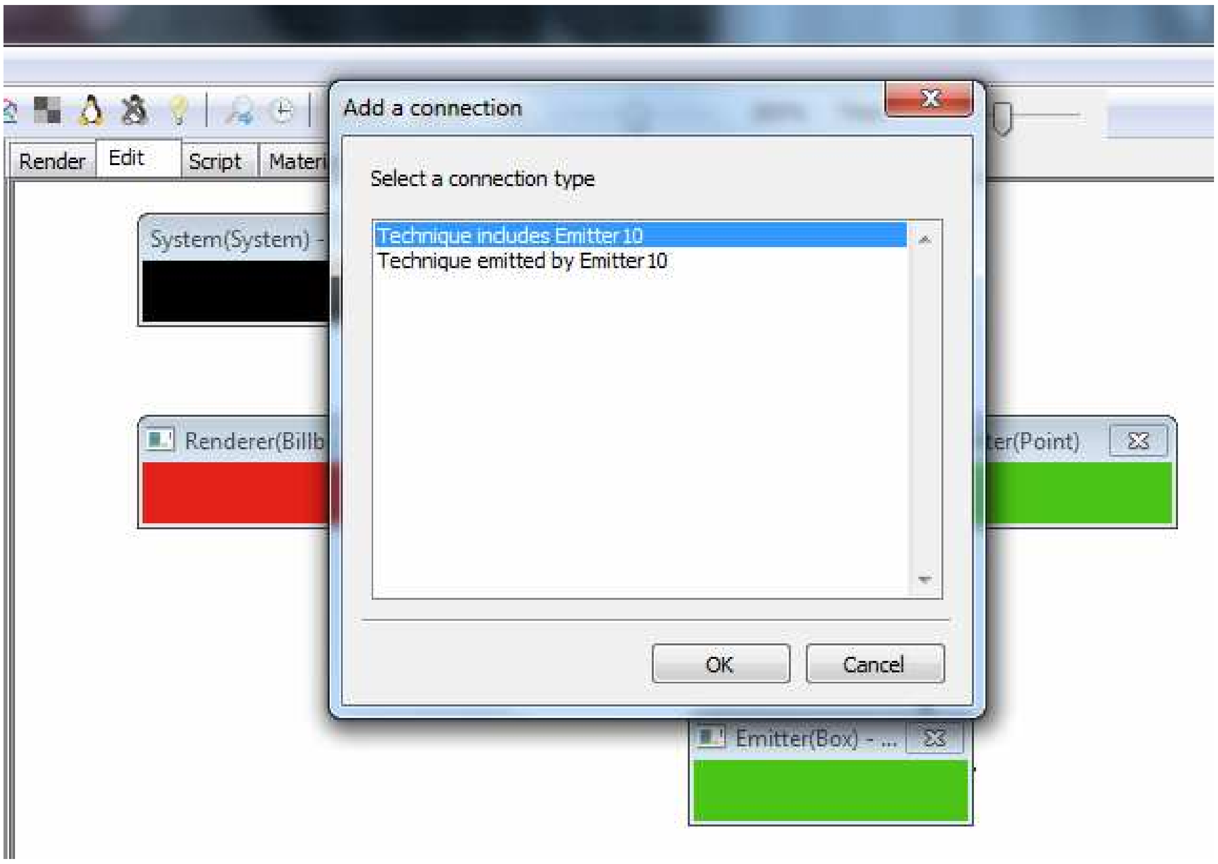
Save the *Lessons_2* particle script.

Lesson 3 - Emission of non-visual particles

Besides emission of visual particles, an Emitter can also emit other Emitters, Techniques and Affectors (only applicable for Affectors for which the position property has a purpose). The plugin also supports emission of other *Systems*, but this is not recommended (performance and memory reasons); therefore it is not supported by the editor.

There are a few example scripts that demonstrate the emission of non-visual particles. Particle system '*canOfWorms*' is a typical example.

Creation of a *System* with emitted Emitters is easy. Create a new System and rename it into *Lesson_3*. Set the **Category** to *Lessons*. Add a Box Emitter. Select the icon and connect the Box Emitter to the Technique.
When you connect the Emitter to the Technique (using the icon), a dialog appears. It is possible have two types of connections between these two components.



Add a connection

Select a connection type

Technique includes Emitter 10
Technique emitted by Emitter 10

Choose the ´includes´ connection.

The connection order (connect Technique with Emitter or connect Emitter with Technique) is not important. The text in the Connection type dialog is from a point of view of the last selected component

The Technique is connected with two Emitters, a Point Emitter and a Box Emitter. Name the Point Emitter *Emitted* (this is the emitter to be emitted) and name the Box Emitter *Base* (this is the base emitter, which emits the Point Emitters).

Establish a connection between the *Base* and the *Emitted* Emitter. A pop-up displays, with 2 possible connection types, either the *Base* emits the *Emitted* or the *Emitted* emits the *Base*. Two possible connections can be made, each with a different result. We want the *Base* to emit the *Emitted*.

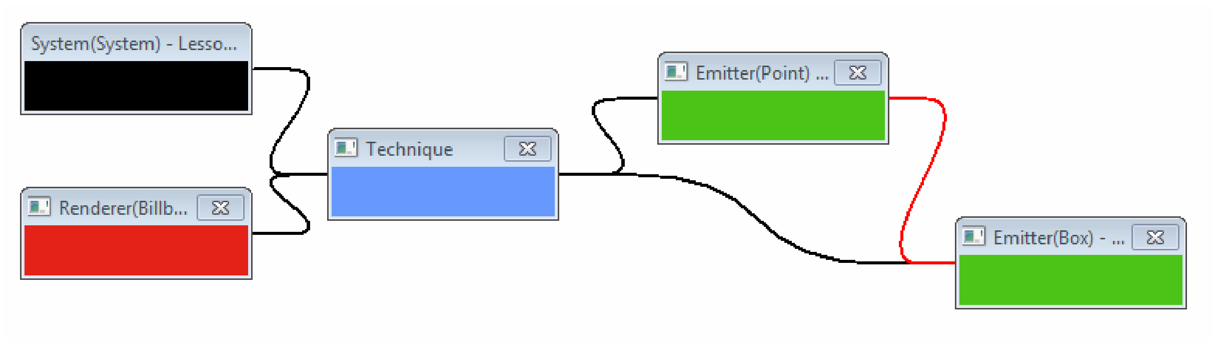
Connecting the line between the two emitters can be done in two directions. The pop-up describes the possible connections in the context of one of the emitters. In the example above this can be:

- *Emitted emits Base*
- *Emitted emitted by Base*

Or

- *Base emits Emitted*
- *Base emitted by Emitted*

The type of connection is represented by a red line:



Fill in some attributes:

- Technique: ***Material*** with value *PUMediaPack/Flare_04*
- Box Emitter *Base*: ***Emission rate*** with a *Fixed* value of *1*

If you start the System, you see that the emitted Emitters emit a group of flares.

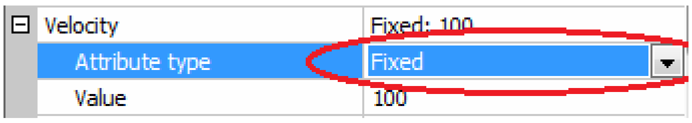
A word of caution: If you are emitting components that include emitters, which on their turn emit other components, the number of pre-allocated objects can become very large and consumes a lot of memory. Use the ***..._quota*** *properties in the Technique to manage this.*

Save the particle system script as *Lesson_3.pu*

Lesson 4 - Dynamic properties

For several properties, it is possible to use a function to calculate the value. One example is the ***Velocity*** property of an Emitter.

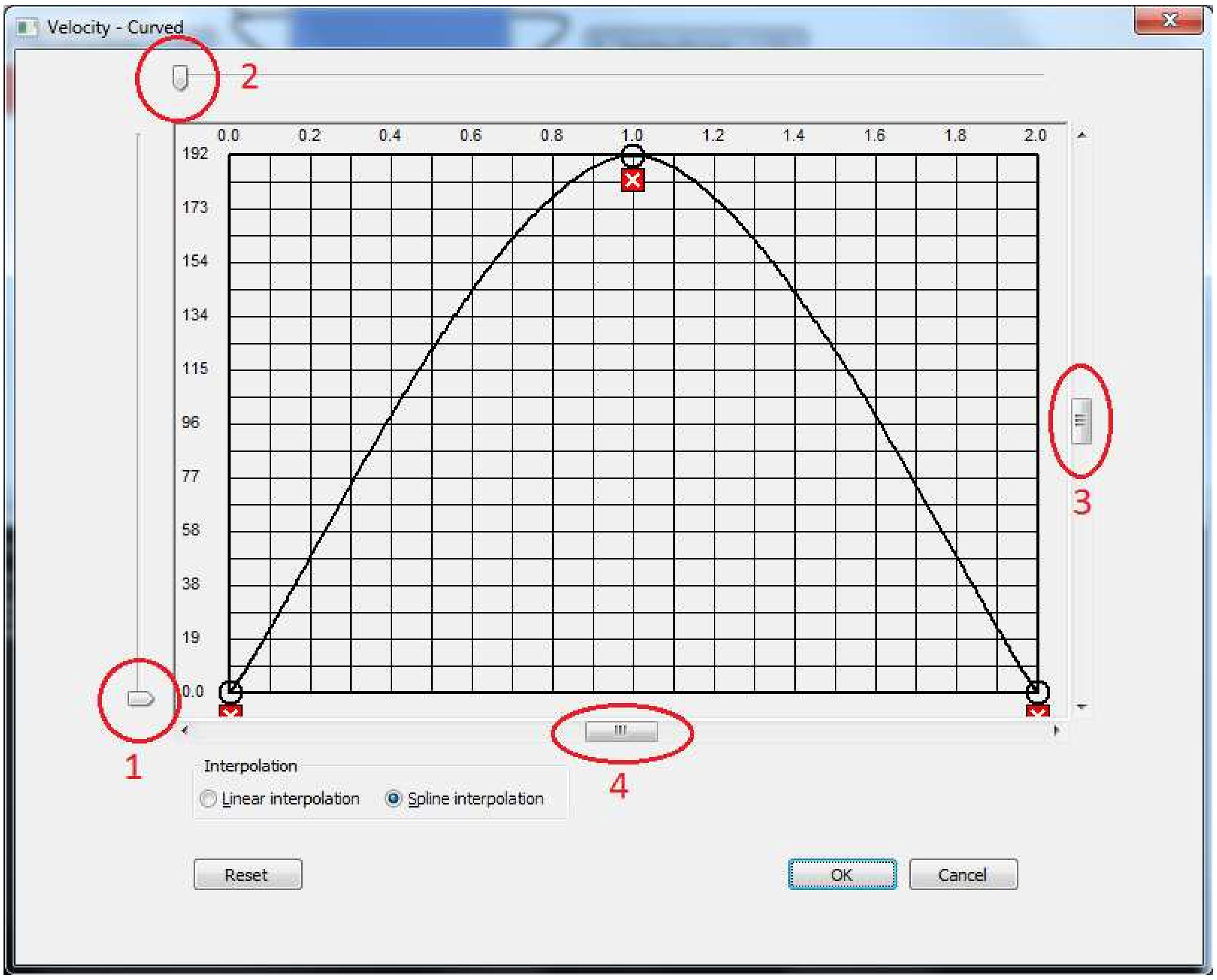
Create a new particle system and assign a nice looking material. Click on the Point Emitter in the Edit-tab. Go to the property window and select the ***Velocity*** property. Expand it by pressing the ⊞ icon. Change the ***Attribute type*** from *Fixed* to *Curved*.



⊟ Velocity	Fixed: 100
Attribute type	Fixed
Value	100

A dialog is displayed. Add three new control points by left-clicking with the mouse on the dialog. Move the control points so that they have the following values:

- Control point left: 0.0; 0.0
- Control point mid: 1.0; 192
- Control point right: 2.0; 0.0



The control points can be moved over the dialog by means of the mouse; set the mousecursor in the ´circle´ of the control point, hold the left mousebutton pressed and move the control point over the window. Also use the two scale sliders and two scrollbars:

- Slider 1: Scale the y value.
- Slider 2: Scale the x value.
- Scrollbar 3: Move the y-axis.
- Scrollbar 4: Move the x-axis.

Control points can be removed by clicking on the ✖ icon, attached to the control point.

Set the checkbox to ***Spline interpolation***. Click on the OK button and restart the particle system. On time 0.0 sec. the particle velocity is 0.0. On time 1.0 sec. after start of the particle system, the velocity is 192. On time 2.0 sec. after start the velocity decreases to 0.0 again.

Lesson 5 - Observers and Event Handlers

Besides emitting particles and affecting them, Particle Universe also has capabilities to observe particles and perform certain actions if a particular event occurs. The Observers included in the Particle Universe Editor package aren't restricted to observing particles only. The OnRandom Observer for example, validates if a generated random value exceeds a predefined threshold.

An Observer must be included into a Technique. Each Observer may include multiple Handlers, which perform an action if the event occurs.

Take a look at example_008. The Technique includes an OnQuota Observer. This means that as soon as the quota of emitted (visual) particles has been reached, the Handler is activated. The (Visual particle) quota is a value defined in the Technique. In the example, the ***Visual particle quota*** it is set to *200*, which means that the Handler is activated if 200 visual particles have been emitted.

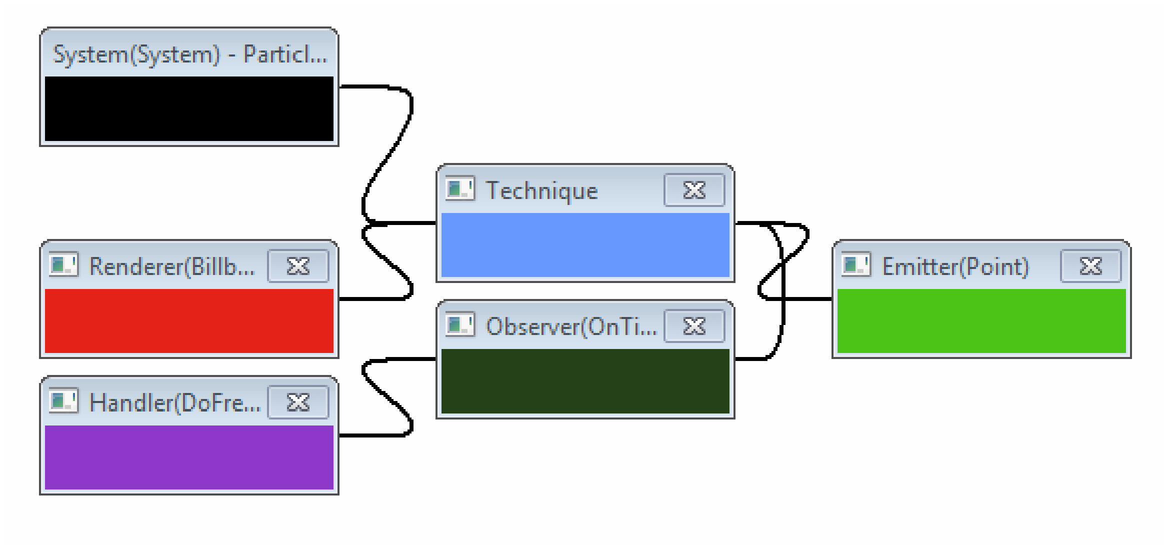
The Handler is of type DoEnable Component and disables the Emitter when the event occurs. Right click on the DoEnable Component to look at the connections. It is 'included' in the OnQuota Observer and has an 'enables' connection with the Emitter. The ***Enable component*** property is set to *False*.

In this lesson we create a particle system with an Observer and a Handler from scratch.

Create a new particle system. Name it *Lesson_5* and set **Category** to *Lessons*. Add an OnTime Observer to the Technique and add a DoFreeze Handler to the OnTime Observer. The icons from the floating toolbar are 👁 for the Observer and 🔧 for the Handler.

Set the ***Material*** in the Technique to *ParticleUniverse/GreenBall*.
Set ***Angle*** in the Point Emitter to *360*.
Set ***Time to live*** in the Point Emitter to *10*
Set ***On time compare*** in the OnTime Observer to *Greater than*.
Set ***On time threshold*** in the OnTime Observer to *5*.
Set ***Since start system*** in the OnTime Observer to *True*.

The Edit-tab looks something like this:



Start the particle system and wait for at least 5 seconds. What happens?
The green balls are emitted in all directions until the time since the start of the particle system is greater than 5 (seconds). This event triggers the Handler, which freezes all particles.

Lesson 6 - Basic collision

Particle Universe has support for both basic collision and advanced collision. Advanced collision is realised by means of PhysX™, which is supported by Particle Universe (PhysX™ is a library/SDK for physical effects in games; see http://www.nvidia.com/object/physx_new.html). PhysX™ is handled in one of the next lessons.

Basic collision is a build-in feature and can be used in situations where advanced collision is overkill. There are four types of collision:

- ***Plane collision*** - represents a plane shape with which the particles collide. Plane collision is realised by means of the Plane Collider (type of Affector).
- ***Box collision*** - represents a box shape with which the particles collide. Box collision is realised by means of the Box Collider and the Box Collider Extern. The first is a type of Affector, similar to the Plane Collider. The second is an Extern object, with similar characteristics, but with the addition that it can be attached to an Ogre::SceneNode. Both the particle system and the Box Collision Extern can be attached to different nodes, which can be moved independently. Usage of the Box Collision Extern needs some additional coding.
- ***Sphere collision*** - represents a box shape with which the particles collide. Sphere collision is realised by means of the Sphere Collider (type of Affector) and the Sphere Collider Extern (type of Extern).
- ***Interparticle Collision*** – Interparticle collision is where particles collide with themselves. If an Interparticle Collider (type of Affector) is added to a Technique, its particles will collide. Inter particle collision can be a performance killer if not tweaked properly. If 100 particles are emitted, 100 * 100 collision checks must be performed.
  This is where the **Spatial hashing** properties of a Technique are used for.

Spatial Hashing

This is a technique to group objects in a 3D space into cells. Particles that are in one cell are close to each other, so only the particles in that cell should be checked for inter particle collision with each other[4]. The difficulty with these settings is, to get the right cell dimensions (not too many particles in one cell and not to many cells with just a few particles).

To complete the collision features, the availability of the Collision Avoidance Affector is worth mentioning. If this Affector is used, the particles (within 1 Technique) try to avoid each other. Collision avoidance is one of the ingredients of flocking behaviour, which is handled in one of the other lessons.

Take a look at example_017, which demonstrates a simple Plane Collider. Also look at example_019, which includes a Box Collider. As soon as the particles collide with the box, the OnCollision Observer in combination with the DoAffector Handler makes them red.

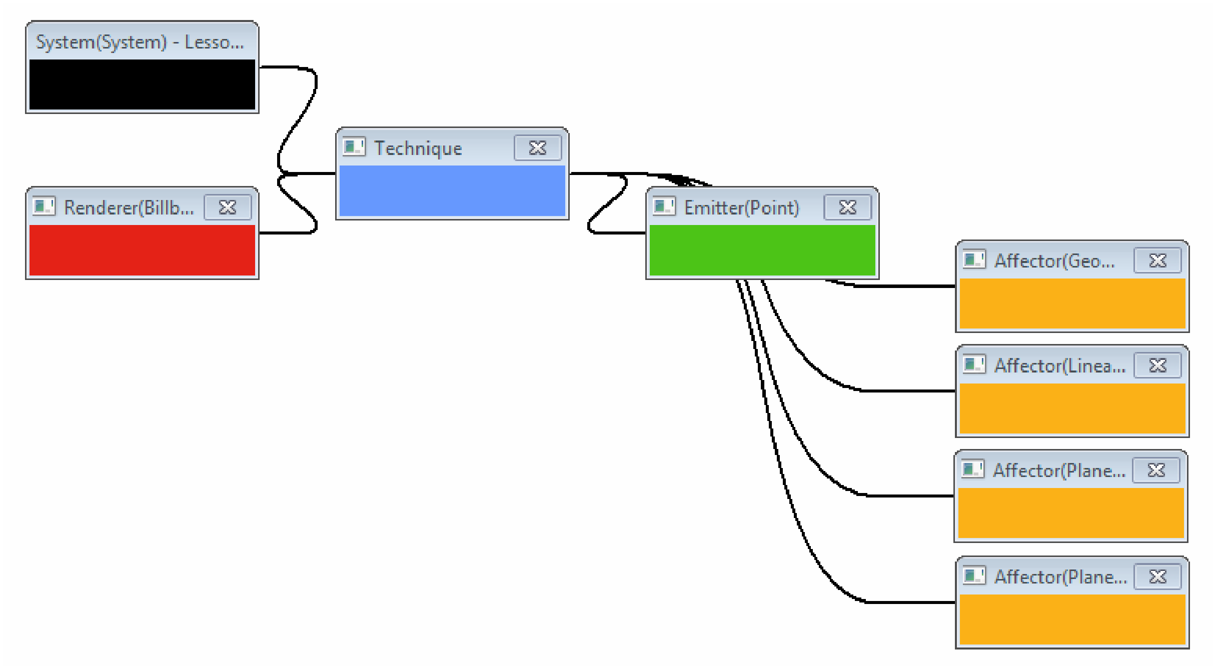
[4] A property called **Spatial hashing cell overlap** can be used to include particles in nearby cells in the collision check.

Let’s create something similar.

Create a new particle system and name it *Lesson_6*. Go to the Edit-tab and change the ***Renderer type*** from *Billboard* to *Sphere*. Change the ***Emitter type*** from *Point* to *Box*. Add some components:

- 1 x GeometryRotator (Affector)
- 1 x LinearForce Affector
- 2 PlaneColliders

Connect everything together according to the image below:



Adjust the settings according to the script:

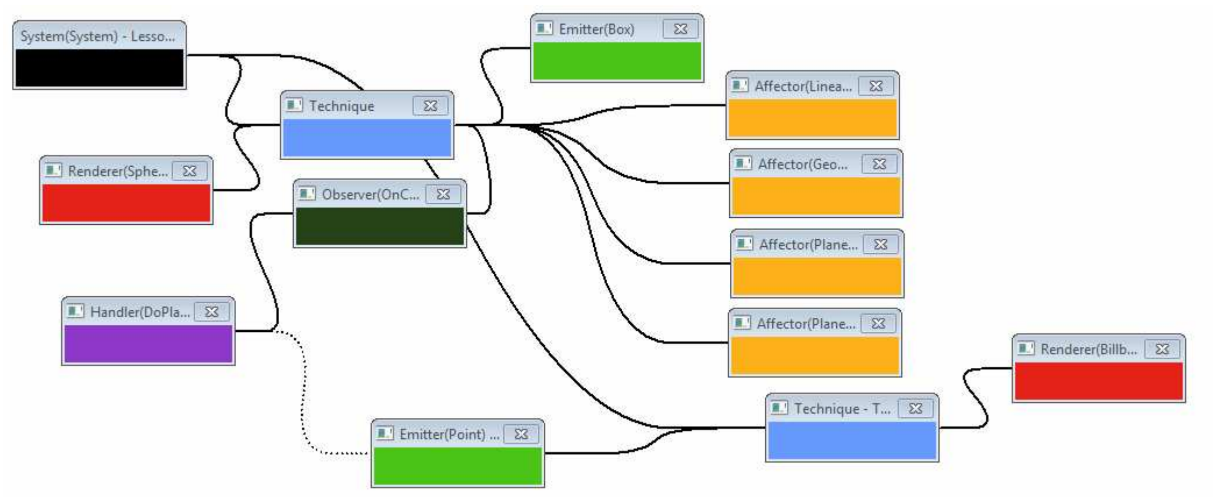
```
system Lesson_6
{
    category                                    Lessons
    technique
    {
        material                                PUMediaPack/BBal
        renderer                                Sphere
        {
        }
        emitter                                 Box
        {
            emission_rate                       3
            time_to_live                        5
            velocity                            800
            direction                           0 0.7 -1
            start_orientation_range              1 1 1 0
            end_orientation_range               1 0 1 1
        }
        affector                                LinearForce
        {
            force_vector                        0 -300 0
        }
        affector                                GeometryRotator
        {
            use_own_rotation                    true
            rotation_axis                       1 1 0
            rotation_speed                      3

        }
        affector                                PlaneCollider Wall
        {
            position                            0 0 -1000
            bouncyness                          0.2
            normal                              1 0 1
        }
        affector                                PlaneCollider Floor
        {
            position                            0 -200 0
            bouncyness                          0.7
            normal                              0 1 0
        }
    }
}
```

Start the particle system; the basketballs are shooting in the Z-direction and hit the Plane Collider with the name *Wall*. They bounce off and hit the Plane Collider with the name *Floor*.

Now, extend the Technique by adding an OnCollision Observer and a DoPlacement Particle Handler.

Also add a new Technique (with ***Material*** is *ParticleUniverse/Flare*), Billboard Renderer and a Point Emitter (***Name*** is *FlareEmitter*) with an ***Emission rate*** of *0,* a ***Time to live*** with value *0.5* and ***Direction*** with value *0,0,0*. Because ***Emission rate*** is 0 means that the Emitter is enabled, but doesn't emit particles by default. Connect the DoPlacement Particle Handler to the Point Emitter with ***Name*** *FlareEmitter*.

After some rearranging, the Edit-tab looks like the image below. Notice the dotted connection between the DoPlacement Particle Handler and the Point Emitter.



The script looks like this:

```
system Lesson_6
{
    Technique
    {
        material                                PUMediaPack/BBal
        renderer                                Sphere
        {
        }
        emitter                                 Box
        {
            emission_rate                       3
            time_to_live                        5
            velocity                            800
            direction                           0 0.7 -1
            start_orientation_range              1 1 1 0
            end_orientation_range               1 0 1 1
        }
        affector                                LinearForce
        {
            force_vector                        0 -300 0
        }
        affector                                GeometryRotator
        {
            use_own_rotation                    true
            rotation_axis                       1 1 0
            rotation_speed                      3

        }
        affector                                PlaneCollider Wall
        {
            position                            0 0 -1500
            bouncyness                          0.2
            normal                              1 0 1
        }
        affector                                PlaneCollider Floor
        {
            position                            0 -200 0
            bouncyness                          0.7
            normal                              0 1 0
        }
        observer                                OnCollision
        {
            handler                             DoPlacementParticle
            {
                force_emitter                   FlareEmitter
            }
        }
    }
    Technique
    {
        material                                ParticleUniverse/Flare
        renderer                                Billboard
        {
        }
        emitter                                 Point FlareEmitter
        {
            emission_rate                       0
            time_to_live                        0.5
            direction                           0 0 0
        }
    }
}
```

When the basketballs hit the Plane Colliders, it is detected by the OnCollision Observer, which activates the DoPlacement Particle Handler, and a flash (created by the emitter with name FlareEmitter) is placed on the contact point.

Lesson 7 - Physics and fluids

Nvidia's PhysX™ can be used to add physical behaviour to Particle Universe particles. Currently there are 2 options:

1. Rigid body based particles: Particle Universe particles are associated with an NxActor. This means that particles derive their values (position, direction, ...) from the NxActor. This is the default behaviour of a PhysX Extern.
2. Fluid based particles: Particle Universe particles are associated with an *NxFluid*.

A particle system with PhysX capabilities can easily be created. Make sure during installation of Particle Universe, that PhysX support has been selected.

Create a new particle system named *Lesson_7* and add an Extern from the floating toolbar. Connect the Extern with the Technique; choose *PhysXActor* as the ***Extern type***. Choose ***Emitter type*** *Box* for the Emitter.

Select a nice material in the Technique component and use the default values. Starting the system, it emits PhysX particles, although the defaults settings don't give a nice result.

Increase the ***Box width***, ***Box depth*** and ***Time to live*** of the Box Emitter and set the ***Y-position*** in the Technique to 500.

> ⓘ
>
> *Note, that a PhysX plane is automatically created in the editor, so that the particles bounce on this plane. The plane settings (and also the gravity) can be changed by means of the PhysX-tab in the Options. Look at the result if these settings are changed.*

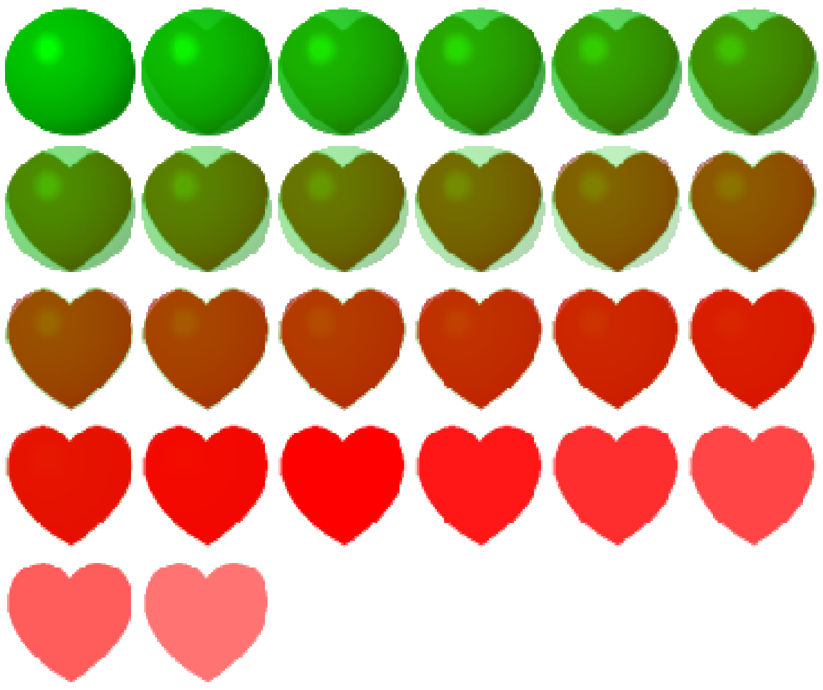
> ⓘ
>
> *The PhysX plane does not need to align with the grid plane that is visual on the sceen; they are two different things.*

Lesson 8 - Texture animation

Atlas textures are supported from version 1.0. Their use is very convenient, because it gives the possibility to assign a different part of the image to each particle individually. In general, the texture of a particle is always identical for particles in a Technique. This is, because only one material can be used in a Technique. However, if you use different uv-sets (the coordinates of a texture) for each particle, the particles appear different.

Defining these uv-coordinates is done in the Renderer by using the properties ***Texture coords rows*** and ***Texture coords columns***. These two properties divide the texture in a matrix of coordinate sets. Each set gets its own index. So, if ***Texture coords rows*** is set to *4* and ***Texture coords columns*** is set to *4*, the texture is divided in a 4x4 matrix with coordinate sets [0..15].

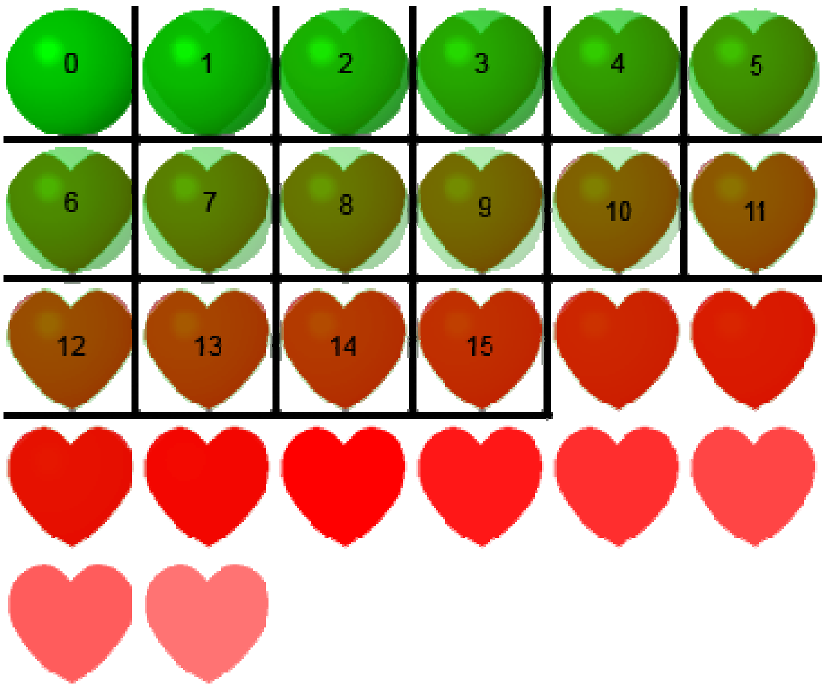
Assume the following texture; it has 6 rows and 6 columns:



A small example (in script format) in which this texture is used (defined in the material 'ParticleUniverse/Interpolate'):

```
system Lesson_8
{
    technique
    {
        emitted_emitter_quota                    10
        material                                 ParticleUniverse/Interpolate
        default_particle_width                   40
        default_particle_height                  40
        renderer                                 Billboard
        {
        texture_coords_rows                     6
        texture_coords_columns                  6
        }
        emitter                                 Point
        {
            angle                               360
            velocity                            200
            direction                           0 0.7 -1
            end_texture_coords_range            15
        }
    }
}
```

The script above emits particles where each particle is represented as a piece of the texture. This piece is defined by properties ***start_texture_coords*** (default = 0) and ***end_texture_coords*** (= 15). The part of the texture is defined as a random texture coordinate between [0..15]. Visualized, these texture coordinates are represented by this figure:



You can take this one step further and add a TextureAnimator. The TextureAnimator is an Affector that displays different uv-sets on a particle as if it was playing a movie. If a TextureAnimator is added to the script above and it is set to loop through the uv-sets [0..16] you see an animation on each particle.
The Particle Universe Editor package contains a script (example_029) that does just that.
To easily create such textures, you can use the AtlasTextureTool, which is added to the Particle Universe package. *The AtlasTextureTool* stitches a number of images together and creates one big image containing all the individual images.

Lesson 9 - Particle System LOD

Particle System LOD (= Level Of Detail) is used to reduce the resource usage of a System so performance will not decrease too much. For example, it is not needed to render Entities (meshes) if the distance between the camera and the particle system is large. The users won't see the difference if particles at a large distance are rendered as Billboards instead.
Important to realise is that with a multiple camera setup the LOD feature doesn't work. The benefit to support multiple camera-LOD seems lesser, because the complexity becomes larger. If the scene has multiple cameras, you also must set the name of the main camera in the particle system (in script this is done with 'main_camera_name').

Particle Universe supports two ways of LOD:

- discrete LOD, making use of multiple Techniques.
- using a ramp function.

Discrete LOD

First, the property ***Lod distances*** must be set in the System. This property controls the distances at which different Techniques can come into effect.
Second, a ***Lod index*** is assigned to each Technique in the System.
The distance at which a LOD level is applied is determined by the ***Lod distances*** property of the containing Particle System. The lay-out of a particle system with three discrete LOD-levels looks like this.

```
system Lesson_9a
{
    smooth_lod                                      true
    main_camera_name                                MainCamera
    lod_distances                                   800 2200
    Technique Lod0
    {
        material                                    BaseWhite
        renderer                                    Billboard
        {
        }
    }
    Technique Lod1
    {
        material                                    BaseWhite
        lod_index                                   1
        renderer                                    Billboard
        {
        }
    }
    Technique Lod2
    {
        material                                    BaseWhite
        lod_index                                   2
        renderer                                    Billboard
        {
        }
    }
}
```

The ***Smooth lod*** property of the System is to apply a smoothness of the transition. As soon as the LOD transition of another Technique comes into effect, the already emitted particles of the previous Technique are still alive and shouldn’t be destroyed immediately. The ***Smooth lod*** property makes sure that these particles still exists even when another Technique is activated. These particles stay alive until they finally expire. The ***Main camera name*** called *MainCamera* is the one that is used to calculate the distance from the Particle System, which is used to execute LOD transitions.

- The Technique with index 0 is activated if the distance between the particle system and the *MainCamera* is between 0-800 units.
- The Technique with index 1 is activated if the distance between the particle system and the *MainCamera* is between 800-2200 units.
- The Technique with index 2 is activated if the distance between the particle system and the *MainCamera* is larger than 2200 units.

Ramp Function

Instead of discrete LOD-levels, it is also possible to influence particular attributes if the camera moves. In the example below, the ***Emission rate*** increases when the camera becomes nearer and decreases when the camera moves away. This is done by means of the ***camera_dependency***.
This can only be defined in a script. The editor doesn't support visual editing of this feature.

```
system Lesson_9b
{
    main_camera_name                            MainCamera
    Technique
    {
        material                                ParticleUniverse/GreenBall
        default_particle_width                  40
        default_particle_height                 40
        renderer                                Billboard
        {
        }
        emitter                                 Box
        {
            emission_rate                       30
            camera_dependency                   emission_rate
            {
                distance_threshold              500
                increase                        false
            }
            time_to_live                        dyn_random
            {
                min                             2
                max                             4
            }
            velocity                            120
            direction                           0 -1 0
        }
    }
}
```

Lesson 10 - Complex structures

Flocking

Particle Universe includes some Affectors, which in combination, perform flocking behaviour. Flocking is "*the collective motion of a large number of self-propelled entities*", exhibited by birds, insects and fish. Flocking behaviour consists of three components:
- **Separation** - avoid crowding neighbours (short range repulsion)
- **Alignment** - steer towards average heading of neighbours
- **Cohesion** - steer towards average position of neighbours (long range attraction)

These three components are implemented in Particle Universe by means of the
- CollisionAvoidance Affector (separation)
- VelocityMatching Affector (alignment)
- FlockCentering Affector (cohesion)

example_027 in the Particle Universe Editor package shows how those three Affectors are used in combination. Both the CollisionAvoidance Affector and the VelocityMatching Affector make use of the Spatial Hashtable functionality, so the Spatial Hashing properties have to be set correct in the Technique for optimal performance.

Flip Flop

In some situations a mechanism is needed where multiple forces that change over time are applied to a particle system. Generation of wind is such an example. This can be done by using some kind of perbutation / noise affector, but an alternative is to use a flip flop mechanism instead.
The basic idea is that multiple Affectors are enabled or disabled in case of a certain event. Take a look at the following excerpt from particle system example *snow*.

```
system snow
{
        ...
        affector                                LinearForce WindLeft
        {
            enabled                             false
            forc_vector                         -20 0 0
        }
        affector                                LinearForce WindRight
        {
            enabled                             false
            forc_vector                         20 0 0
        }
        observer                                OnRandom
        {
            observe_interval                    1
            handler                             DoEnableComponent
            {
                enable_component                affector_component WindLeft true
            }
            handler                             DoEnableComponent
            {
                enable_component                affector_component WindRight false
            }
        }
        observer                                OnRandom
        {
            observe_interval                    1
            handler                             DoEnableComponent
            {
                enable_component                affector_component WindRight true
            }
            handler                             DoEnableComponent
            {
                enable_component                affector_component WindLeft false
            }
            random_threshold                    0.6
        }
        ...
    }
    ...
}
```

Depending on a random value, the Linear Force Affector with the name *WindLeft* is enabled (and the *WindRight* is disabled), or vice versa.

Emission chain

Example_024 is a good example of an emission chain. An Emitter from one Technique emits another Technique. The emitted Technique also includes an Emitter that on its turn emits another Technique, etc. This is visualised by the red lines in the Edit-tab.

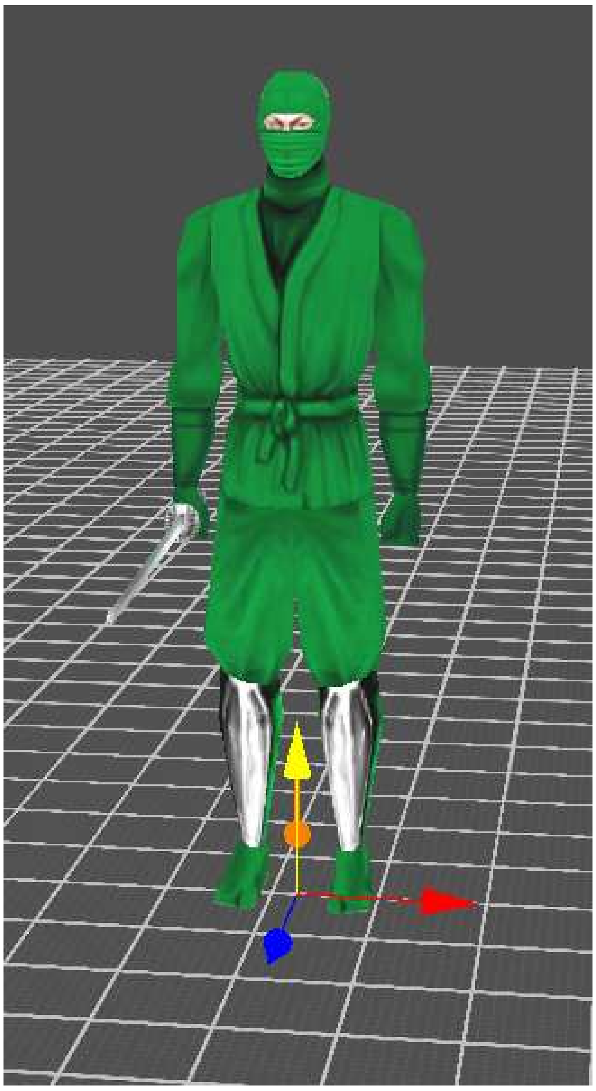
Lesson 11 - Adding a mesh to the scene

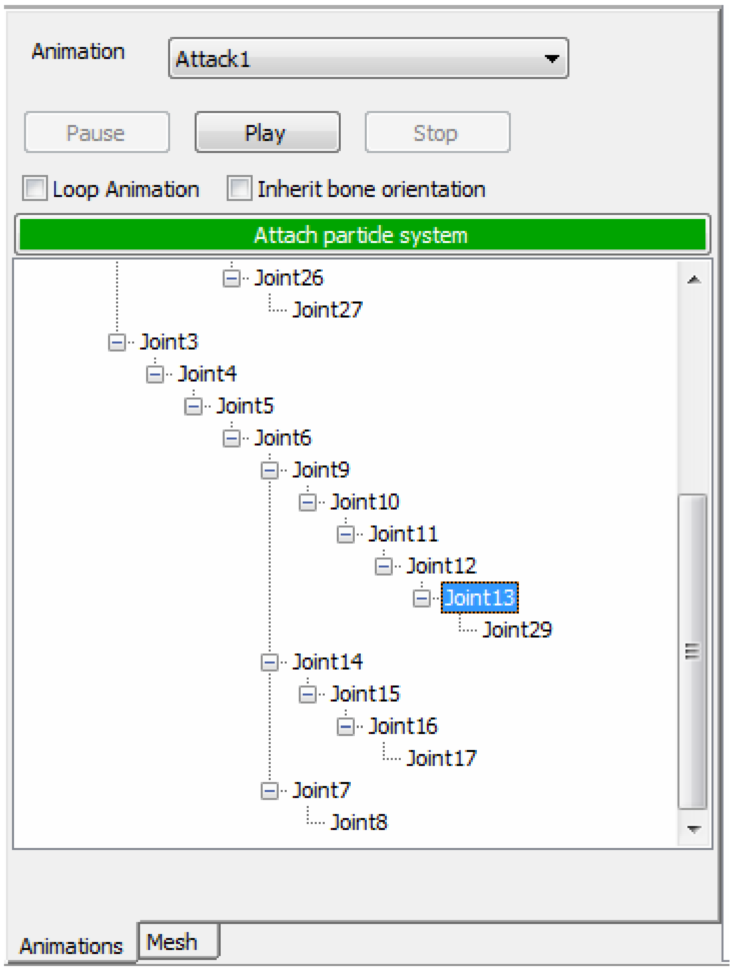
In many games, particle systems are attached to a mesh, which performs some kind of animation. The Particle Universe Editor makes it possible to simulate this behaviour, so particle systems can be created and adjusted in combination with in-game animations.

First, go to the Render-tab and select the particle system *fireSystem* from the treelist. Add a mesh with animation to the scene by dragging and dropping the mesh from the Windows file explorer to the Render tab of the editor. The example used is the animated **ninja.mesh** from the Ogre package (this is not included in the Particle Universe package!). After the **ninja.mesh** is dropped to the Render tab, additional resource locations must be selected from the directory dialog that pops up.

> *Of course it is also possible to add an (animated) mesh by means of the mesh listbox that is displayed when the icon is pressed. The resource location(s) where the mesh and its underlying components (material, skeleton) are stored, must be set of course.*

The ninja mesh is displayed in the scene and on the left its skeleton structure is displayed.

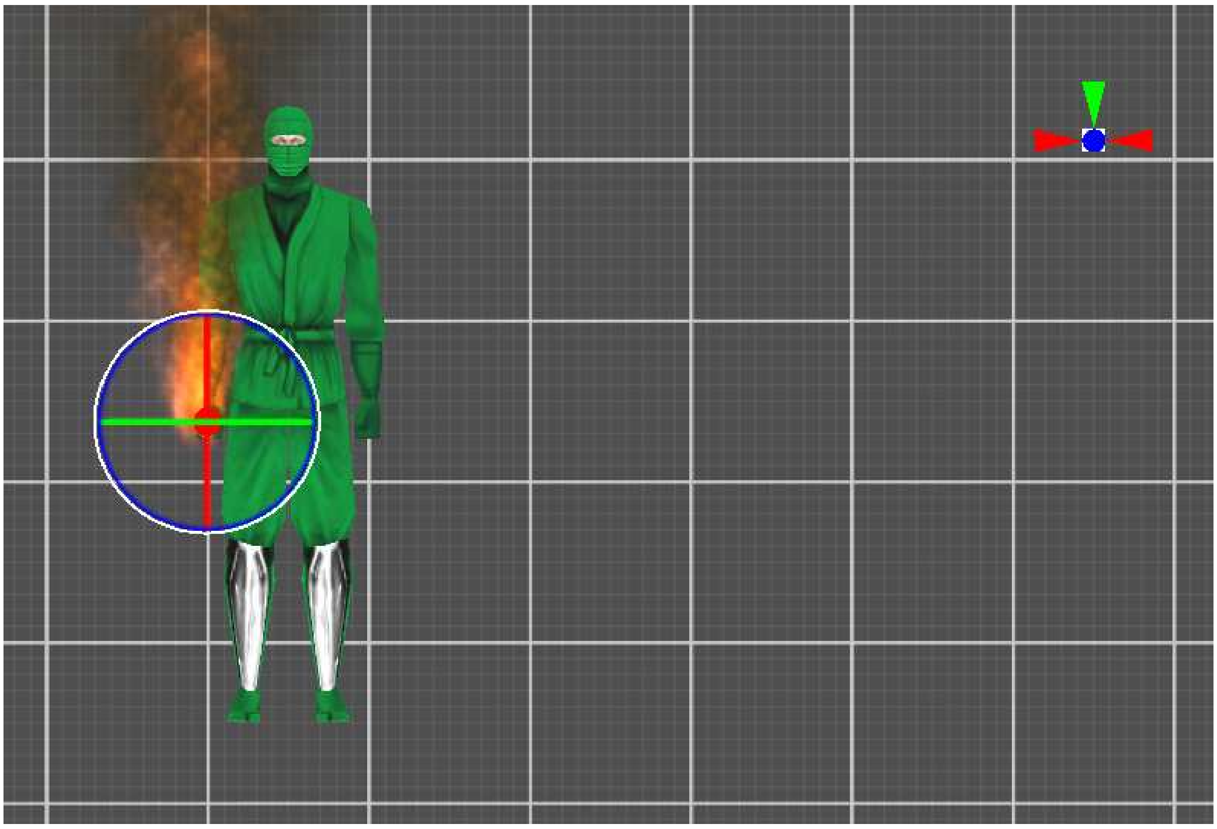




Select the particle system called *fireSystem* from the tree list with particle systems. Go to the Edit tab and set ***scaleVelocity***, ***Scale dimensions.x***, ***Scale dimensions.y*** and ***Scale dimensions.z*** all to *15*.
Go back to the Render tab and select a Skeleton from the skeleton tree on the left (i.e. Joint 13). A green button with the text **Attach particle system** appears. Press this button to attach the particle system to the bone.

Press the **Play** button to start the animation (selected in the dropdown listbox). As you can see the flames orientation is pointing in the Y-direction. To align the flames with the sword, select the particle system by pressing either the TAB or 'Q' key, or select with the mouse.

Select the Rotation Gizmo . Sometimes it is easier to align in orthographic camera mode. Press the icon and choose for the perpendicular view by means of the icon. The screen looks tike this (if the particle system is running):



Align the flame with the sword by means of the gizmo and play the animation again.

When playing the animation with an attached particle system, you can choose whether the particle system inherits the bone orientation (checkbox on the left side of the screen. If it is checked, the particle system keeps following the orientation of the bone to which it is attached. If unchecked, the initial orientation of the particle system remains the same, although it still inherits the orientation of the mesh itself.

To have a better idea of the synchronisation between the animation and the particle system, use the time slider in the toolbar to decrease time and run the animation and particle system in slow motion.

Lesson 12 - Recording

Recording is a function that produces a series of image files (png, jpg, tiff) or a video file (avi / swf / wmv). In the Quick Tour the configuration screen with the Record tab has already been discussed.

The output can be used in at least three situations:

- The images form a sequence of the particle system in action. It is possible to use them in-game, as a type of 'particle system imposter'. If you create (in code) a BillboardSet with just 1 Billboard, a material can be applied that includes the series of images and the animation type.
  (See http://www.ogre3d.org/docs/manual/manual_17.html#SEC79)
- An alternative to this, is to create an Atlas texture from the images with the Atlas Texture Tool in the Particle Universe package. Create a System that emits 1 Billboard particle in combination with the TextureAnimator.
- Use the images or video files in video editing software. This goes beyond this manual, but several editing packages support import of video files or a series of image files. The recorded particle system can be mixed with other videos.