



How to Use X-Particles

By Steve Pedler

Contents

Contents	2
Foreword	6
Disclaimer	6
Version and copyright	7
About the Author	7
How to Use this Book	8
Part 1: Getting Started	10
Chapter 1: Basic use	10
1.1 Particles and their data	10
1.2 Modifiers	14
Summary	16
Chapter 2: Groups	17
2.1 What is a particle group?	17
2.2 Using groups	18
2.3 Changing a particle's group	22
Chapter 3: The Emitter: Emitting Particles	23
3.1 How are particles created?	23
3.2 Creating particles	25
3.3 Different emitter shapes	31
3.4 Defined emission	31
Summary	32
Chapter 4: The Emitter: Emitting from an Object	33
4.1 Changing to object emission	33
4.2 Where to emit from	33
4.3 Other options	41
4.4 Topology	45
4.5 Emitter falloff	49
Chapter 5: The Emitter: Extended Particle Data	51
5.1 Particle rotation	51
5.2 Other data	56
5.3 Physical Data	57
5.4 Fluid Data	57
5.5 Custom Data	57
Chapter 6: The Emitter: Particle Display	60
6.1 Showing the particles	60
6.2 The HUD	64
6.3 Modifier fields	64
Chapter 7: The Emitter: Other Tabs	67
7.1 Groups tab	67
7.2 Questions tab	67
7.3 Modifiers tab	67
7.4 Editing tab	67
7.5 Advanced tab	69
Chapter 8: Rendering particles	70

8.1 The X-Particles material	70
8.2 Rendering particles with Cycles 4D	75
Summary	75
Part 2: The Middle Ground	77
Chapter 9: The Generator Object	77
9.1 Generating objects	77
9.2 Generating multiple objects	77
9.3 Generating animated objects	78
9.4 Other settings	78
9.5 Object morphing	80
9.6 Modifiers affecting the Generator	84
9.7 Generator tags	84
Summary	85
Chapter 10: The Sprite Object	86
10.1 Using the Sprite object	86
10.2 Text sprites	87
10.3 Rubble sprites	88
10.4 Sprite textures	88
10.5 The Sprite Shader	91
10.6 Sprite modifiers	92
Chapter 11: The Trail Object	95
11.1 Generating trails	95
11.2 Trail thickness and colour	102
11.3 Other options	103
11.4 Rendering the trails	104
11.5 Other objects affecting the trails	106
Chapter 12: Questions	110
12.1 Basic principles	110
12.2 Question types	113
12.3 Combining questions	125
12.4 Question order	126
12.5 What happens when a question passes?	126
Summary	127
Chapter 13: Actions	128
13.1 Basic principles	128
13.2 Types of Action	130
13.3 Action objects: common interface elements	130
13.4 How-To: use actions to turn modifiers on and off as required	132
13.5 'Other' Actions	135
13.6 Control Modifier and Dynamic actions	137
13.7 Object actions	138
13.8 Direct actions	140
Chapter 14: The System object	145
14.1 Creating a System object	145
14.2 System object functions	146
Chapter 15: Modifiers Part 1	150

15.1 Basic principles	150
Chapter 16: Modifiers Part 2: Controller Modifiers	156
16.1 Blend modifier	156
16.2 Change Group modifier	157
16.3 Color modifier	159
16.4 Custom Data modifier	163
16.5 Freeze modifier	164
16.6 History modifier	165
16.7 Infectio modifier	167
16.8 Inherit modifier	171
16.9 Kill modifier	174
16.10 Particle Life modifier	175
16.11 Negate modifier	177
16.12 Physical modifier	178
16.13 Python modifier	180
16.14 Scale modifier	181
16.15 Sound modifier	183
16.16 Trails modifier	190
16.17 Transform modifier	192
16.18 Trigger Action modifier	193
16.19 Unlink TP modifier	194
16.20 Weight modifier	195
Chapter 17: Modifiers Part 3: Motion Modifiers	198
17.1 Attractor modifier	198
17.2 Avoid modifier	200
17.3 Cover/Target modifier	203
17.4 Direction modifier	210
17.5 Drag modifier	214

(This page intentionally left blank)

Foreword

X-Particles (hereafter 'XP') is a huge and powerful particle system for Cinema 4D. It has grown way beyond its original creators' intentions and visions for it. There is no other particle engine available for C4D that comes even remotely close in terms of power, speed and capabilities.

But with such size and power comes complexity. It is now so big that it can be intimidating for new users. Although there is a comprehensive reference manual and a video manual, there is no guide for new users or any single place where a user can just...find out how to do stuff.

This is the purpose of this book. I prefer books to videos since if you need to go back and look something up, it's much harder in a video than in a book. This book is intended to be:

- A means to get you started with XP
- An explanation, without being highly technical, of how things work internally so that you can better understand how to use it
- A way to show you how to do things with XP, both simple and more complex

The book is not:

- A replacement for the reference or video manuals
- A series of long tutorials
- Intended for expert users of X-Particles
- Completely comprehensive - no book could encompass every single thing you can do with XP

There are two ways of writing a book. The first is to write all the content, polish it up, format it as required and publish it in some way. Typically for a book like this such a process would take months, probably a year or even two. By which time of course, it is out of date - the screenshots don't match the latest version, more features have been added and some possibly removed. That reduces the usefulness of the book considerably.

The other way is to write a section and make the updated book available as new sections are added. This ensures that it is as up to date as possible and that you, the user of XP, can get your hands on it immediately rather than waiting months for a new edition. That is the way I have decided to write this book.

This means that the book will never actually be finished. There will always be more to add and corrections to make. Even supposedly complete sections will have new content added to them. The only way to do this is to publish the book electronically, but I don't want to make it available only online: you might want to read it when travelling or when you don't have internet access. Therefore, the book will be published as a single PDF file you can download. If there is sufficient demand I may also make it available as a MOBI file for the Kindle e-reader, but that is only a thought at this stage.

Disclaimer

It's important that you understand that this is NOT an official publication by INSYDIUM LTD. It is entirely a personal venture. Please do NOT approach INSYDIUM with comments or suggestions for the book, they won't be able to answer or to add or change anything in it.

If you have any such comments please send them to me using the contact form on the book's website.

However, I cannot answer queries about how to do various things with XP; answering those would eventually be a full-time occupation! The best way to get answers to such queries is in the Discord channel for XP and other INSYDIUM products. I am, of course, always willing to hear ideas for future content for the book, but I can't promise to add any of them - it all depends on time.

Version and copyright

'How to Use X-Particles' and associated scene files, animations, plugins and add-on items (if any) are copyright Steve Pedler, 2021, 2022.

The current version is 1.4, released April 21st 2023.

While copies may be freely distributed to anyone who wants one, alteration of the contents or attribution is not permitted. In other words, please don't change the text or the images in it. You are of course allowed and indeed encouraged to play with and alter the various scene files as much as you like!

Please do not host this book and/or its associated files on another website, as that only leads to out-of-date versions circulating on the internet. Feel free, however, to link to the book's website so that anyone who needs a copy can obtain the latest version. The book will always be free of charge so there is no danger that it will suddenly only be available at a cost to you, the user of X-Particles.

The book's website URL is <https://xpbook.microbion.co.uk>.

Last but certainly not least the author acknowledges and is very grateful to Mike Batchelor for the artwork used on the book's cover.

About the Author

Why am I writing this book? I was the sole developer for the first version of XP released in 2012 (though the original idea for it was not mine) and as it grew and the company expanded I remained in a senior developer position until early 2021.

This is the book I always wanted to write to accompany XP, but what with other things - you know, stuff like actually coding the software and writing the reference manual - there was never time to do it. Now I have that time, and this is the result.

This book is, and will always remain, free of charge. I don't intend to make money from it. Although I'm no longer working to develop XP and the proportion of my code in it will inevitably reduce over time, I still regard XP as my baby. This book is, if you will, a labour of love.

Steve Pedler
August 2022

How to Use this Book

The intention of this book is to flesh out areas which are not covered in the manuals in detail because of space limitations or because a reference manual is just that - a reference, not a 'how-to' type of book.

This is intended to be a how-to style of book so that new users can find out to how to do some of the common things you might want to do with X-Particles. If you want to know how to do something, turn to the relevant section and hopefully you will find what you need. If you don't, use the contact form on the book's website and I may be able to add something in a future revision.

In addition to the text, I have supplied a number of scene files to show how things are done. They are in an archive for each chapter which you can download from the [book's website](#). There are also animations which show the results of various actions; these aren't embedded in the text because they add hugely to the file size so you will find links in the book to the animation on the website (or you can just watch them all there).

Please note that for the interface screenshots in this book I used Cinema 4D R25 on Windows, so it may look a little different from the interface in earlier versions of Cinema 4D. The build of X-Particles is the latest build available (which was 1260 at the time of writing). All example files were made in C4D R19 and are compatible with all C4D versions from R19 onwards.

The scene files have deliberately been kept short and simple. There is no elaboration or fancy features in them - they exist simply to demonstrate a particular point in the use of XP and so that you don't have to set up the file before trying the feature out. Don't expect detailed, high-quality artwork from them; they really are absolute bare-bones files. For more detailed and very functional scenes, I would recommend the content repository and the formal training videos on the INSIDIUM website (see <https://insydium.ltd/support-home/>).

Finally, where necessary I may include add-ons for XP which let you do something which XP can't normally do, or possibly just as an extra for anyone who wants them.

New in version 1.4, April 18th 2023





The first five 'motion modifiers' are covered in this version. In addition, there is an important change to the way elements in the user interface are shown in the text. Previously, a colour field in the interface of an object would have been shown like this: 'Color'. Although this book is not intended as a reference manual, it's clear that it would be helpful if readers could find details about a particular interface element more easily. For this reason, all such interface elements (only for X-Particles objects) will be shown in bold red text like this: '**Color**'.

The same thing has been done to the names of example files, so they can be seen more readily. Formerly, the names of example files looked like this: *an_example_file.c4d*. Now they look like this: ***an_example_file.c4d***.

If you find any elements of file names which are not highlighted in this way, please do let me know.

Other notes

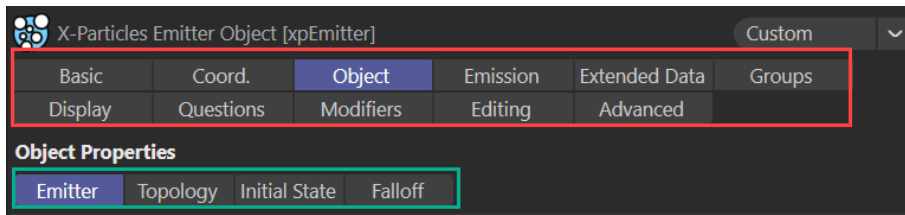
In the book, in addition to links to files and animations you will see these icons:

-  What follows this icon is a handy tip which might save some time and frustration in getting something to work.
-  These are important points to remember when using a particular feature.
-  You see this icon when there is a file to download (usually an archive file containing the example scene files for a chapter).
-  This icon shows that an animation referred to in the text is visible on the book's website.

There are some conventions used in this book. To avoid confusion the ones to note are:

- The spelling used in this book is British English. The user interface for XP, however, uses American English. This can occasionally lead to what might look like typos: 'color' in the interface vs. 'colour' in the text, for example. These aren't typos, they are me being pedantic.

- The frame rate for all the example scenes is 30 frames per second. Bear in mind that when you read that something will be carried out 'in 30 frames' this really means 'in 1 second' which would be in 25 frames if the project frame rate was 25 frames per second.
- A 'tick box' in the user interface I call a 'switch' because that's what it is, it turns something on or off. If there is a tick in the box the switch is 'enabled' so to enable or turn on a switch you click it so it shows a tick mark (or check mark if you prefer). A disabled switch is turned off and has no tick/check mark.
- Cinema 4D's interface has the facility to show settings grouped in tabs. In the XP interface many tabs also have their own sub-tabs which we call quicktabs. They are shown here in part of the emitter's interface. The tabs are contained in the red box and the quicktabs in the green box:



Part 1: Getting Started

Chapter 1: Basic use

This chapter covers the very basics - what particles are, what an emitter is, what modifiers do and so on. You can skip this or just skim the headings if you are already familiar with X-Particles or other particle systems. But it is highly recommended for users new to a modern particle system!

1.1 Particles and their data

This section is an overview of what particles are and the data they carry.

↓ The scene file(s) for this chapter can be found in the archive [examples_ch1.zip](#) on the book's website. Click here to [download the archive](#).

1.1.1 What are particles?

Perhaps more importantly, what are they not? Particles are not objects in a scene, like the primitive objects or splines, or generators such as the Cloner. Individual particles don't appear in the object manager and they don't have points or polygons. So what are they?

A single particle is simply a collection of data. All particles possess a set of basic data; some may have additional or extended data items. By changing this data you can alter the behaviour and/or appearance of the particle.

One particle isn't too much use, so typically you would have many more - perhaps millions. Internally they are stored in a particle array, which is just a block of memory containing the data structures for each particle. You don't need to access this array though, unless you are writing an add-on for XP.

How do you create them?

This is the most basic operation in XP and so of course it uses the most complex object! To create particles is the job of an XP Emitter, which is discussed in detail in chapters 3 through 7.

What can you do with them?

Particles were originally developed to enable fluid simulations. By having many tiny particles move in the scene rather than trying to move a vertex in an object, far more fluid-like appearances became possible and the particle behaviour approaches that of a real fluid. By 'fluids' we mean not only liquids but other things that behave in a fluid-like fashion, such as gas, smoke and flame.

You can do all those things with XP but it can do a lot more. XP uses particles for cloth simulations and to move the shards of a shattered piece of geometry. Each particle can have an object associated with it, so that moving the particle moves the object and deleting the particle removes the object from the scene. You can use particles as the basis for a polygon mesh, using the XP implementation of OpenVDB. Then at its most basic, you can even render the particles themselves without any scene objects to produce things such as sparks, or to produce abstract artwork.

1.1.2 Basic particle data

All particles have a set of basic data. At its most elementary a particle really needs to know only two things: where it is and where it is going. So the basic data includes the particle position in the 3D world. To determine where it goes next (the next time it is moved, that is to say, in the next frame) it also has a velocity, which comprises speed and direction. That will determine how far it moves and in which direction.

That, in itself, is quite powerful and you can do a lot with just those data items. But there are many more parameters which are used so frequently that they are included in the basic data for all particles. These include:

- radius
- colour
- shape
- lifespan
- index and ID values
- group ID number
- ...and lots more!

Some of these need further explanation.

Radius

A particle has no size, it is infinitely small. But that isn't much use if we want to apply the size of a particle to an object linked to the particle, for example, or if particles are to collide with a scene object. For that reason, a particle is given a virtual size. Each particle is then considered to be a sphere (even if it has a different shape on screen, it is still a sphere for internal purposes) and has a radius in scene units. Typically, individual particles have a small radius of 1-3 units, compared to (say) a primitive Sphere object with a radius of 100 units.

Note that the radius may have a profound effect on whatever simulation you are developing - you may sometimes need a large radius and sometimes a very small one.

Scale

Particle scale isn't very useful unless you are generating geometry to attach to a particle. As an example, suppose we have an emitter with an XP Generator linked to it, and the Generator has a Cube primitive as a child object. When the playback starts, the cubes will be sized according to the particle radius. so if we set that to 3 +/- 2 units, we see this with lots of different-sized cubes:

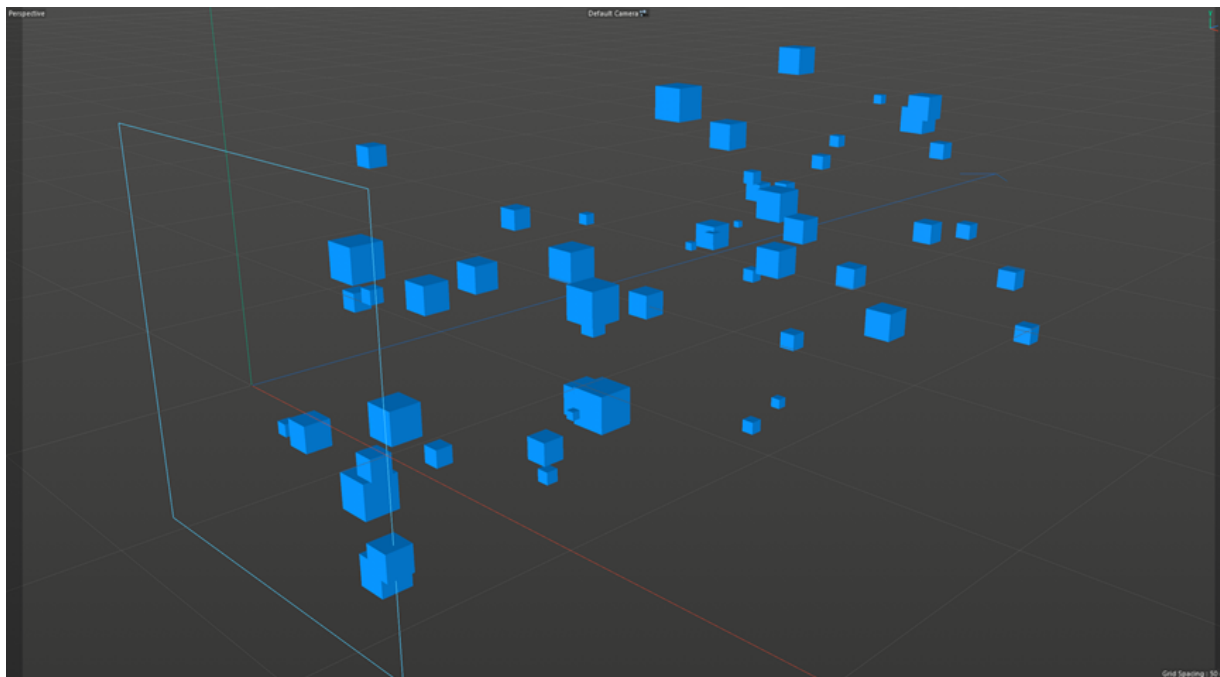


Figure 1.1. Particle radius and scale

But the cubes are small - much smaller than the default cube. How does the Generator do this? By default, the scale of the generated object is derived from the particle radius and when the objects are generated they are given this scale. You can see this if you generate some objects, do Current State to Object on the Generator, and look at the scale of the polygon objects which result from that. Notice that it's actually very small - the scale in this example will be 0.03 +/- 0.02.

This works well, but it always gives a uniform scale to the objects. Let's imagine that you'd like the cubes to be three times longer on the Z axis than the original object. You can't do this by changing the radius, but you can change the particle scale. By default the particle scale is set to 1, but if we leave it like that and set the object scale from the particle scale, we'll get full-size cubes, so we need to reduce it to a suitable value so we get something like the above result.

The first thing to do is disable the **'Uniform Scale'** switch. This is a convenience switch so that changing the scale on one axis automatically changes it on the other two. We don't want that because we want the Z-axis scale to be different. Now we can set the scale to 0.02, 0.02, and 0.06 on the respective axes. This will give similar sized cubes to the screenshot above. We can also add in some variation; so we'll leave 'Scale Var.' set to a uniform value and simply enter 0.015 into one of the axes fields.

As things stand this won't do anything because the Generator is set to use the particle radius to control object scale. We need to change that in the Generator's 'Scale, Rotation and Offset' tab so that **'Scale Using'** is set to **'Particle Scale'**. On playback we now get this:

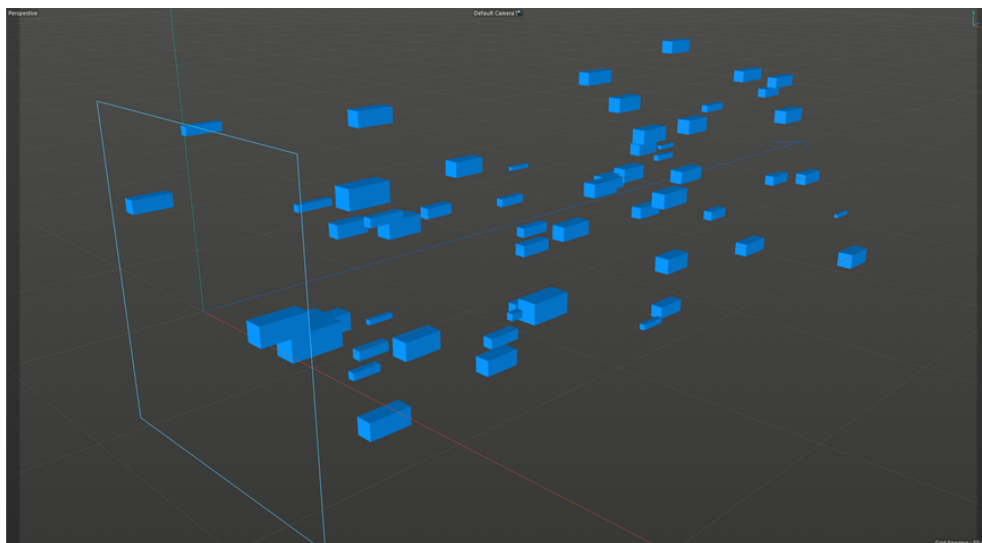


Figure 1.2. Using non-uniform scale

This scene file is in the download archive as *ch1_particle_scale.c4d*.

Colour

This is the colour used to draw the particle in the editor. However, the colour may have other effects. For example, if you generate actual objects from the particle using the Generator or Sprite objects, the generated object will have the same colour as the particle associated with it.

Shape

The particle has to have some kind of shape so you can see it in the editor. Although each particle is treated as a sphere, it is possible to draw them as different shapes in the editor. There are several reasons for doing this:

- you want to use a shape which shows the particle radius - the default Dot shape won't get bigger if the radius increases, so if you need to see that, you might choose a Circle or Sphere shape instead
- you want to use a shape which responds to the particle scale. Not all shapes will change if the scale changes, so again, if you need to see that, you could choose a Box shape
- you want to render the shape in the final scene; rather than generating a separate object from the particle you could use a shape such as Arrow and render it directly using XP's Display Render object
- you have a lot of particles and want the viewport to be drawn as fast as possible - some shapes take a lot longer to draw than others

i See the reference manual for a useful table showing which shapes change with changes in radius or scale. In summary though, it works like this:

Shapes not affected by particle radius or scale:

- Dots
- Ticks
- Squares
- Lines (the line length is greater the higher the particle speed)

Shapes affected by radius but not by scale:

- Axes
- Spheres
- Circle
- Circle (Filled)
- Arrow
- Arrow (Filled)

Shapes affected by scale but not by radius:


- None, there are no shapes affected only by scale

Shapes affected by both radius and scale:

- Box
- Box (Filled)
- Cylinder
- Pyramid
- Plane
- Plane (Filled)

When it comes to speed, dots and squares are the fastest to draw. The cylinder and (perhaps surprisingly) pyramids are the slowest.

When choosing a shape other than Circle or Sphere, always remember that internally the particle is still considered to be a sphere. This is very important when using the Collider tag to make the particle collide with a scene object. The collision detection function assumes a spherical shape for the particle so choosing a shape such as Arrow may look as if the collision isn't working correctly - but it is, if you remember that the internal shape is always a sphere.

 The above lists are only for the appearance of the particle in the viewport. Even if a particle shape isn't affected by scale, if you change the scale you may not see any difference in the viewport but the scale will still affect any other aspect of XP which uses it. In other words, the scale value internally is whatever you set it to be, even if this doesn't alter the size of the particle shape in the viewport.

Lifespan

Each particle has a set lifespan and an age. The age starts at zero and when it exceeds the lifespan, the particle will die and is removed from the scene. You can set the lifespan when the particle is created and then alter it using a particle modifier, and can also kill a particle immediately, regardless of its remaining lifespan.

Index and ID values

As mentioned above, the data structure for each particle is stored in an array. It is possible to iterate through all the particles using the index into the array for each one. However, this won't allow you to do something to a specific particle because the index may change.

Consider this scenario. You have 10 particles in a scene each of which have an index value. This is zero-based, so the first particle in the array has index 0 and the last has index 9. Now suppose in the next frame the particle at index 0 is killed (deleted from the scene) and another one is added, which will have an index of 10. Unfortunately you now have a block of memory at index 0 which is not being used for anything - the particle has died, so its data is no longer needed. This is very inefficient, and if you have tens of thousands of particles with constant deletion of particles and others being added, you could end up wasting a huge amount of memory.

For this reason, when a particle is deleted and/or new ones added the array is rationalised to remove unused space. Deleting particles but not adding enough to fill all the empty slots will cause the array to shrink, possibly with many particles ending up with a different index.

Therefore, you cannot use the index value to point to a specific particle. To enable that, each particle is given a unique ID value which is guaranteed never to change. This starts at 1, and increments as each particle is created. If a particle is deleted its ID value will

not be reused. Again, in the above scenario the original 10 particles will have ID values from 1 to 10. Deleting the particle at index 0 and creating a new particle will give the new particle the ID value 11.

Group ID number

The group ID number determines which group a particle is in. All particles are in a group; unless you change it, this is the default group which has the ID number zero.

See [Chapter 2 \('Groups'\)](#) for more details of what groups are and how you can use them.

1.1.3 Extended data

Although each particle carries its own set of basic data, in many cases additional data parameters are required for specific functions. For example, if you want to rotate particles, a set of rotation data is added.

Why isn't such data always available? It's a question of memory. Each particle takes up a block of data for its basic data. The more that is added to that data, the more memory is used. This is very wasteful for data items that you might never need. For example, if you aren't generating smoke or fire, you don't need the physical data such as temperature, etc.

In some cases you need to add the extended data required manually. For example, if you want to be able to rotate particles you need to enable rotation by checking the switch **'Use Rotation'** in the emitter. In many cases however the required extended data is added automatically. If you use a Cover/Target modifier for example, the modifier itself will add the extra data it needs. In most cases you can't (and don't need to) access this data, but occasionally some is made available to the Question object, the Xpresso nodes, or when using data mapping.

1.2 Modifiers

The emitter is an essential component of X-Particles, but there are many other objects to use. These are all optional, but you will almost always use one or more of them in your scene. In fact, if you do anything other than just generate particles, you will need some other component. The most frequently used of these are the particle modifiers.

1.2.1 What are modifiers?

Particle modifiers do exactly as their name suggests: they change something in the particle. Most modifiers do only that, but some can also generate additional particles or other structures.

X-Particles categorises all its modifiers into four groups. These are:

1. Controller modifiers which change particle data in some way
2. Motion modifiers which change particle motion (i.e. speed and direction)
3. Sprite modifiers designed to work specifically with XP's Sprite object
4. Generator modifiers which generate more particles or other structures

There is sometimes a considerable overlap between these categories, but it's a convenient way to divide them up.

1.2.2 How modifiers work

When you add a modifier to a scene, by default it will work on every particle from every emitter in the scene. You can change this if required but that doesn't matter for now.

Each frame, the emitter works through a cycle of functions. It builds a list of all the modifiers which apply to this emitter, then calls each modifier in turn to do its work on the particles in the emitter's particle array. For each particle in the array, the modifier determines if it should do anything to that particle and if so, it carries out whatever function it is intended to do.

Note that whatever change is made, this won't be something that only occurs once, in a single frame. If you use a Speed modifier with its default settings, it will increase particle speed by the specified amount every frame and will not stop doing so unless the particle is no longer affected by the modifier for some reason. Even if you use a modifier with what looks like a single change, for example a Speed modifier set to **'Absolute'** mode, which sets the speed to a specific value, that effect is repeated each frame, so if something else causes the speed to change the Speed modifier will simply change it back again. There is a caveat here though. What

you actually see will depend on what order the modifiers work in and when other processes are carried out. For example, if you have a Speed modifier set to **'Incremental'** mode, the speed will increase each frame. If you now add a Question to the emitter which triggers an Action to set the speed to a specific value, the speed won't increase at all. The reason is that the Speed modifier increments the speed, but then the Question/Action pair are executed and set the speed to a specific value. Because Questions are executed after any modifiers, the Speed modifier appears to have no effect.

Once all the modifiers have finished, there will be some further work to be done by the emitter. For example the Kill modifier may have marked a particle for deletion, so the emitter must remove that particle. The emitter will also move particles along their direction with their given speed, both of which may have been changed by one or more modifiers.

Determining which modifiers work on which emitters

Each emitter builds its own list of modifiers which will work on it. By default, every modifier in the scene will be included in that list and will therefore work on every emitter. But modifiers can be excluded from any emitter, in the following ways:

- a modifier may be added to an emitter object's Modifiers list; if that modifier is set to be excluded from the emitter, it won't work on any particles from that emitter
- the emitter may be set to use only modifiers which in the object manager are in the hierarchy of the same System object (if any) as the emitter

Determining which particles the modifier will affect

Just because a modifier is added to the emitter's list of modifiers to operate on that emitter doesn't mean the modifier will affect all, or indeed any, of the particles from that emitter. For each particle, the modifier will act on it if:

- the particle is in the modifier's field of effect, which it always will be unless a Field object (or Falloff in versions of c4D prior to R20) is added to the modifier and the particle is outside the field/falloff area
- the particle is in a particle Group which is affected by the modifier (see [Chapter 2](#) on Groups for more details)
- the modifier is in Independent mode, or is in Action-Controlled mode and has been activated for the particle under consideration
- in some cases, if the particle satisfies some other condition set in the modifier itself

Only then will the modifier actually do something to the particle.

This system gives you very fine control over which particles are affected by a modifier, right down to the level of specific individual particles, if desired.

Modifier modes

As indicated above, each modifier can have one of two modes, independent or action-controlled. There is a very significant difference between them.

In independent mode, the modifier works on all particles from all emitters, subject to the conditions listed above. You don't need to do anything other than add the modifier to the scene. That's fine, but suppose you want the modifier to work on certain particles - let's say, you want to change the particle colour but only if the speed exceeds a certain value.

To do that, you need action-controlled mode. In this mode, the modifier will not work on any particle at all, until it has been activated to work on specific particles - in this case, those whose speed exceeds the required value. This is done by using an Action object, which when invoked will activate the modifier.

It works something like this if we think about the above scenario. Each frame, a Question object is used to test the particle speed. If the speed is higher than the required value, an Action object is invoked by the Question object and this will set a flag internal to the particle which specifies that from then on a specific modifier will work on that particle. Other particles which don't pass the Question test will not be affected by the modifier. From then on the particle will always be affected by the modifier as long as that internal flag is set. If the flag is cleared - which another Question could do - the particle won't be affected by the modifier any further.

Questions and Actions are discussed further in chapter 12 and 13 of this book.

1.2.3 Modifier types

Controller modifiers

This is a large group of modifiers which carry out a number of very different functions. The majority are there to change some aspect of the particle data - size, colour, group ID value, and so on, but not specifically the speed or direction of travel which are the purview of the motion group (although some controller modifiers do affect these parameters as well).

Some controller modifiers are a good deal more complex, such as the Infectio, History and Python modifiers, while others are there to do a single specific job, such as the Unlink TP and Weight modifiers.

Motion modifiers

As the name implies, the primary function of these modifiers is to affect particle motion. They do this by changing the direction of travel and/or the speed. In some cases this is very simple, for example, the Speed modifier just changes the speed without altering direction, while the Direction modifier does the exact opposite. Many, however, will change both - sometimes quite radically.

Sprite modifiers

These are a set of modifiers which work specifically with the XP Sprite object. This object, and the modifiers which affect it, will be covered in a later chapter.

Generator modifiers

Five of these modifiers actually cause more particles to be generated, either from the same emitter which invokes the modifier or from another emitter. The other two act on generated geometry so are used specifically with the Generator object.

Summary

X-Particles has a lot of modifiers - 54 in the current build I'm using for this book. Some of them, such as Turbulence, you will probably use a lot. Some of them you may never use, but it's worth taking some time to explore the reference and video manuals to see what they can do. You may find that you can do something really interesting with an obscure modifier you've never used.

Chapter 2: Groups

Groups are a very important aspect of X-Particles. Among other things they let you determine which modifiers will work on a particle, but they have many more uses than that. Let's start by looking at what groups are and how a particle is assigned to a group.

2.1 What is a particle group?

All particles, without exception, belong to a group. If you haven't assigned a particle to a group when it is created, or done anything to change it after creation, the particle will belong to the default group.

A particle can only belong to one group. This can be changed - a particle can be moved from one group to another - but the particle can't be added to multiple groups at the same time. If you need to do that, you can't use the standard group but you can use *metagroups* instead; these are discussed in the Metagroups section later on.

A particle group is a number. That's all it is. Each additional group has its own unique number and this number is stored in the particle so it 'knows' which group it is in. These numbers increment as you add more Group objects to the scene. The default group has the number zero; if you add a Group object, this will have the ID number 1, the next one added will have ID 2, and so on.

You don't have to add a Group object to the scene to add particles to the default group; unless a new particle is to be assigned to a different group, they will all go into the default group, whose ID number is zero. Note that you cannot create a new group with an ID of zero; that is a special group that always exists and you cannot delete it or create a group with that number.

You can probably see from this that if you have more than one emitter in the scene, all the particles are in the same group (zero) by default. This is very important to understand: a group does not have to be unique to particles from a single emitter. You can create multiple groups and assign particles from different emitters to the same group, if you wish. Or you can make a group specific to one emitter so that only particles created by that emitter will be in the group.

It is not possible to edit the default group directly because it is really a place holder in case you want to assign particles to another group. There is no Group object for the default group and as mentioned above, you can't create one. For all other groups, you must create a Group object before you can assign particles to it.

2.1.1 The Group object

The interface for this object isn't complicated and is shown in Figure 2.1. If you look at the object's interface, you can see that this is a sort of combination of the emitter's Emission, Display, and (for the physical data) Extended Data tabs. Most of the parameters are identical to the emitter parameters of the same name and will be covered in the next chapter. However, there are two which need some further explanation.

Group Number

When you create a new group X-Particles will give it a unique number, which will be one higher than the highest group number already in the scene. You can edit this but it is strongly recommended that you don't do so. If you do, you must ensure that all the groups have a unique number. Otherwise you could have two or more groups with the same number, and this number is used for a number of things, including deciding if a modifier is going to have any effect on a particular group.

Mode

There are three entries in this menu. The default is **'Standard'**, in which case the Group parameters will determine the display and emission settings for any particle which is assigned to this group when the particle is created. This is going to be what you want most of the time. However, sometimes you will want the group also to determine the physical settings such as mass, temperature and so on. These are the parameters found in the emitter's Extended Data tab, **'Physical'** quicktab. If you want the group to determine these, you must set this menu to **'Extended'** and then you can edit the settings in the **'Physical'** quicktab in the Group object.

Conversely, you might want the group to control only the viewport display and not the emission settings such as speed, leaving those to the emitter. In that case, change this menu to **'Display Only'** and you will only be able to alter the display settings.

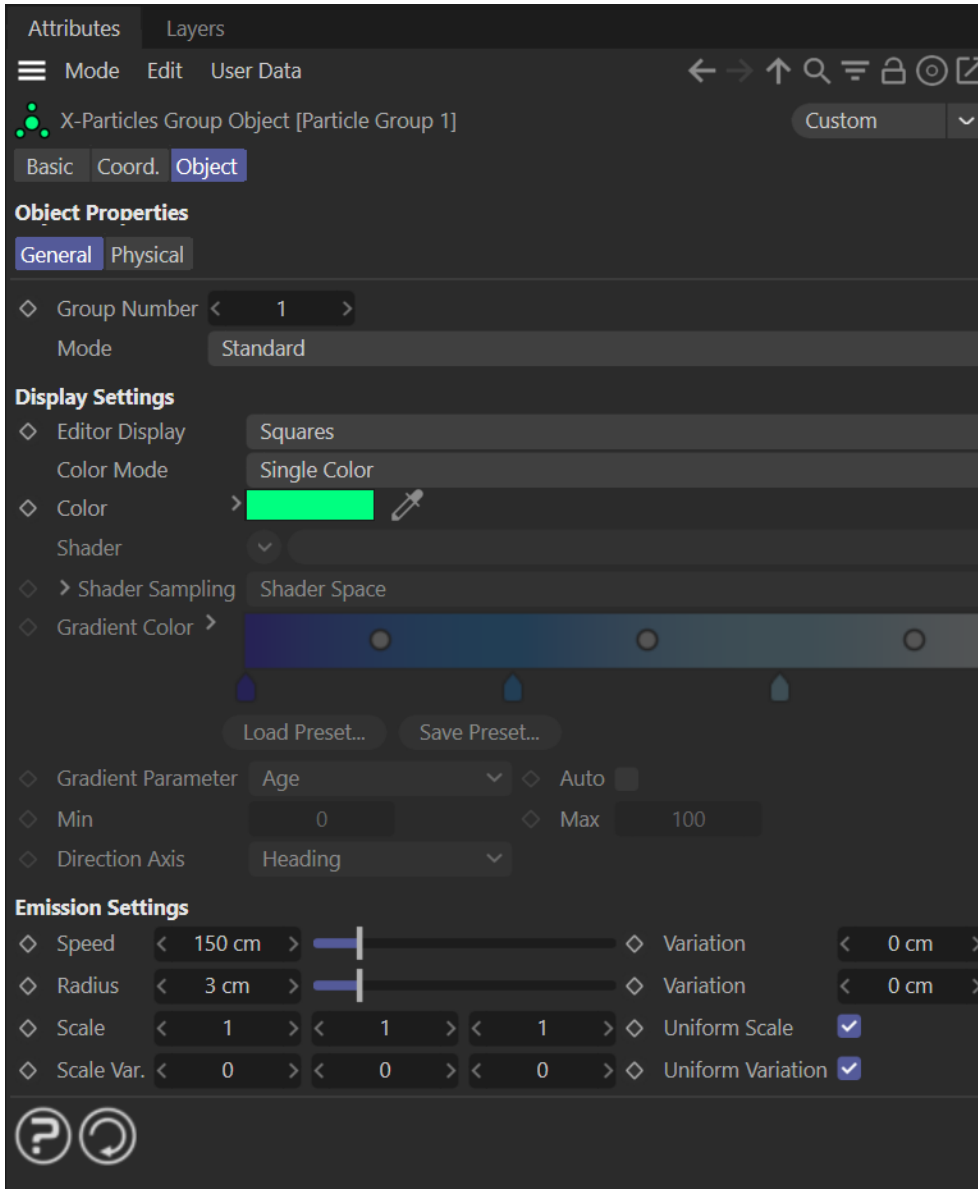


Figure 2.1. Group Object interface

2.1.2 Assigning particles to a group

Particles are assigned to a group when they are created, although you can change the group later (see below). To have an emitter put the particles it creates into a group, the Group object must be present in the **'Groups to Use'** list which is found in the **'Groups'** tab of the emitter.

What happens if there is more than one group in the list? This is discussed in the next section, 'Using Groups'.

2.2 Using groups

Why bother with creating groups and adding particles to them? There are a number of things you can do. You can use a group to:

- emit particles with different characteristics from the same emitter
- ensure modifiers only work on a specific subset of particles
- test the group ID in a Question object and trigger an action depending on the result
- have Actions work only on particles in some groups but not others
- make only particles in specific groups collide with objects in the scene

This can give you a lot of control over sets of particles. Let's take a look at some examples.

2.2.1 Giving particles from the same emitter different characteristics

As a very simple example, suppose you would like half the particles from an emitter to be red circles with a radius of 5 units, and the other half to be blue boxes with a radius of 2 units. There is more than one way to do this. You could:

- emit all the particles with the same parameters then use a combination of Questions and Actions to set these characteristics after creation
- or use Xpresso to do the same thing
- or use an XP Python modifier

But it's much easier to use a couple of groups instead. To do this we need to create two new groups, which we do by adding two Group objects to the scene. You can do this in one of two ways. If you add the Group object from the main XP menu, then when created the group will have the same parameters as the default values in an emitter when that is created. You will have then to drag and drop the Group object into the '**Groups to Use**' list in the emitter.

However, in the '**Groups**' tab in the emitter you see a button labelled '**Create and Add Group**'. There is also a switch, '**Use Current Particle Settings**'. If the switch is enabled and the button clicked, a Group object will be created with the same current parameters as the emitter, even if some of those had been changed before the Group object was created. If the switch is disabled the group is created with the default parameters. Whatever the state of the switch, clicking the button will create a Group object and add it automatically to the list; it will also take you straight to the new object so you can edit it (if you don't want that to happen, hold down the Ctrl key when clicking the button).

Why create two groups? Can't we create one and split the particles between the default group and the new group? It doesn't work like that. If there are no groups in the emitter's '**Groups to Use**' list, all particles go into the default group. But if there is at least one group in the list, all particles go into those group(s) and none go into the default group.

We can now edit the group characteristics to change the shapes, colours and radius values. The Groups tab will look like this:

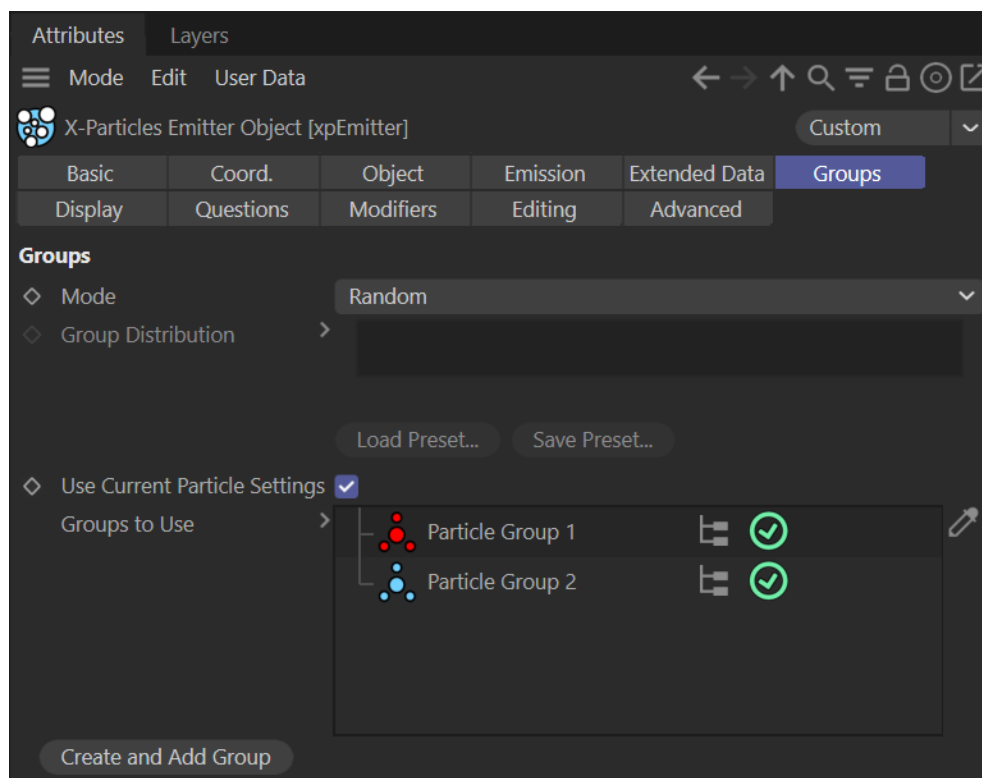


Figure 2.2. Using multiple particle groups

Emitting some particles will show this working as seen in this screenshot (Figure 2.3):

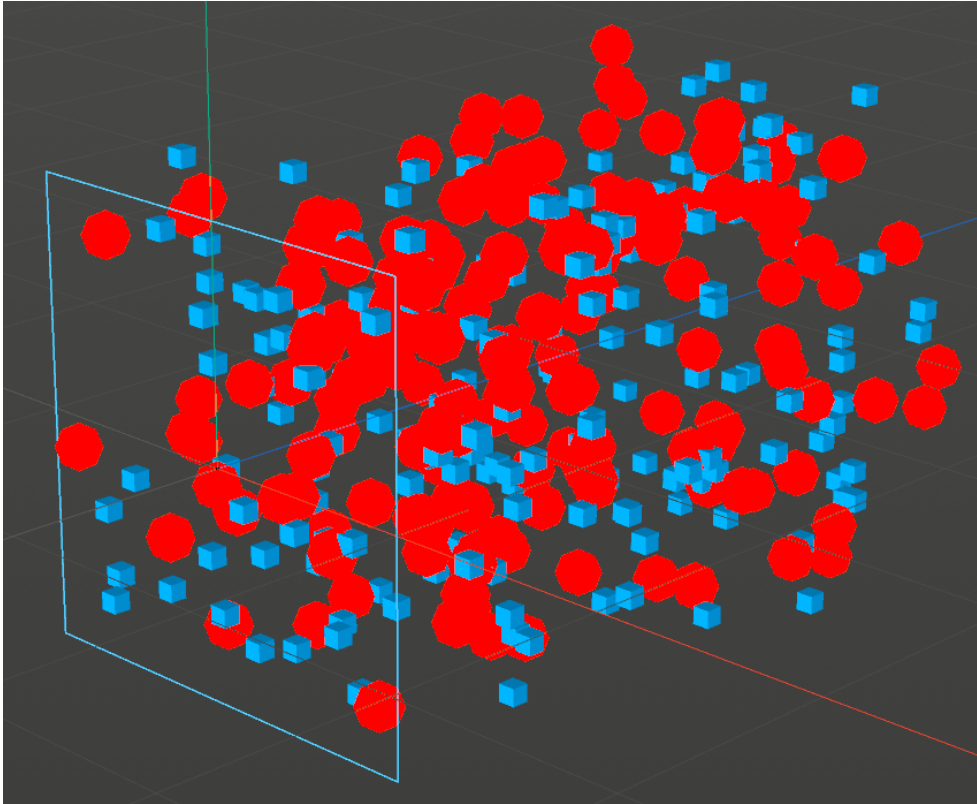


Figure 2.3. Effect of using different groups

You can see from this that half the particles are in the red group and half in the blue. That's the default - particles are assigned randomly to a group, so the distribution is roughly equal between the two. There are several distribution modes available which are listed in the X-Particles reference manual, but one needs a little more explanation.

2.2.2 Controlling distribution between groups using a gradient

At present there is no way to allocate particles between groups with numeric values (e.g. 20% in group 1, 30% in group 2, and so on). You can do this using the gradient, however.

If you select **'Use Gradient'** in the **'Mode'** menu in the emitter's **'Groups'** tab, a gradient control becomes available. The colours used in this gradient have no effect whatsoever. The important point is the position of the gradient knots. At first there are no knots, so all the particles will go into one group, which will be the last group in the **'Groups to Use'** list. By adding one knot we are effectively dividing the gradient into two parts. The first part determines how many particles go into the first group in the list, while all the remaining particles go into the second group. If more groups are added, more knots will be needed. Three groups need two knots, which divides the gradient into three parts. You can, of course, specify the position of each knot precisely. To show how this works, let's add a third group which shows the particles as yellow arrows. Then we'll arrange the gradient knots like so:

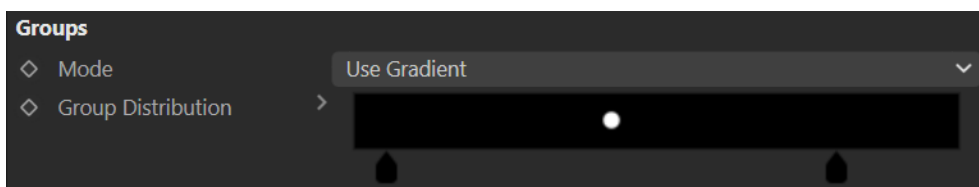


Figure 2.4. Group distribution using a gradient

The first knot is at 5%, the second at 80%. Playing the animation is shown on a screenshot in Figure 2.5 on the next page.

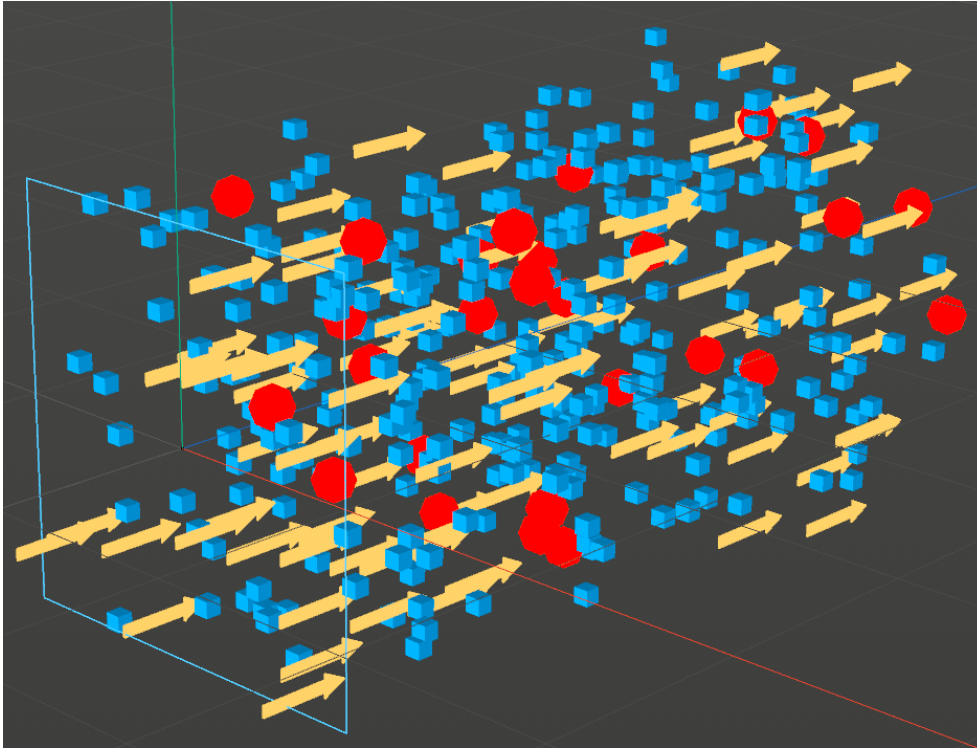


Figure 2.5. Distribution of particle groups using the gradient in Figure 2.4

What is happening is that the red circles are the first group in the list, so 5% of the particles go into this group. The blue cubes are the second group so 75% go into this group (the first knot is at 5%, the second at 80%, so the gap between them is 75% of the gradient's length). The yellow arrows are the third group so the remaining 20% of particles are added to this group.

To make this work correctly the number of knots should be the number of groups minus 1. If you have too few knots, some groups won't appear at all. If you have too many knots, only the knots which are needed will be used, so if there are three groups but 10 knots, only the first and second knots will be used. In practice, this means that all the groups would appear in the scene but particles in the last group in the list will be over-represented.

2.2.3 Use groups to ensure modifiers work on a specific subset of particles

This is straightforward. Simply drag the Group to be affected by a modifier into the '**Groups Affected**' list of any modifier which has one (most do). Then the modifier will only work on particles belonging to the group(s) in the list.

2.2.4 Using groups with Questions and Actions

You can test if a particle is in a group (or if it is not in a group) by adding a Question object to an emitter and changing the '**Data**' menu to '**Group**'. Then drag the group, or groups, to be tested into the '**Groups**' list of the Question object. If you add an Action to the '**Actions**' list, that Action will only be triggered for particles in that group.

However, there's a catch here. Suppose you have two identical emitters. Both emit particles in the same two groups. You want an action to be triggered for the particles in group 1 but not group 2, so for the first emitter you add a Question to test if a particle is in group 1, then trigger the action if so.

This works fine for the group 1 particles emitted from the first emitter but has no effect on group 1 particles from the second emitter. This is because the question is only tested for the first emitter, not the second. There are two ways round this problem. The first is to drag the Question object into the Questions list of the second emitter, so that the question is now tested against particles from both emitters. This works as expected, but there's another way. Delete the Question objects altogether then drag the Group object for group 1 into the '**Groups Affected**' list of the Action object. Now the action will affect group 1 particles from any emitter. The problem now is that the action isn't being triggered at all, since the Question object has been deleted. We'll look at how to solve that - that is, triggering actions without using questions - in Chapter 13.

2.2.5 Other ways to use groups

You can specify groups in other XP objects. For example, if you add an xpCollider tag to a scene object, you can drag Group objects into its **'Groups Affected'** list to restrict collisions with the object to only the specified particle groups.

Another example would be the Generator object. If you add groups to its **'Groups Affected'** list, only the particle groups in the list will have geometry generated for them. This can be quite useful if you want to turn off the generation of geometry temporarily for certain particles, perhaps to speed up the playback, without disabling the emitter or Generator.

2.3 Changing a particle's group

The group ID is not fixed and unchanging. Remember, it is only an ID number stored in the particle. You can do that in a number of ways:

- use a Change Group modifier
- use a Change Group action
- use Xpresso with the Set Particle Data node

We'll cover all of these later on.

As you can see, groups have many uses if you need to restrict certain effects from some sets of particles. Though they do only affect groups of particles, not individual particles, so they're something of a blunt instrument. But of course, you can get as granular as you like and affect one specific particle out of a million...if you really need to. To do that means knowing how to use Questions and Actions, which we will look at later on.

Chapter 3: The Emitter: Emitting Particles

This chapter looks at the emitter and how to emit particles in various ways. It also looks at the extended particle data and the display options.

↓ The scene file(s) for this chapter can be found in the archive [examples_cb3.zip](#) on the book's website. Click here to [download the archive](#).

3.1 How are particles created?

To create particles you require an emitter. The emitter is the only way to create particles, so it is the core of everything that XP does. In most cases you will create the emitter (or emitters) that you need manually, but in a few cases it will be created for you and placed into the scene. In some other cases an object may create an emitter to be used internally to that object - these are not placed in the scene and you cannot access them or change their parameters.

3.1.1 Particle emission

When a particle is created the emitter does a number of things:

- the required data structure (position, speed, radius, etc.) is created and initialised with default values
- the structure is added to the particle array
- the emitter then moves the particle from its initial position using the speed and velocity it has been given
- finally, the emitter draws the particle in the scene viewport using its colour and shape parameters

This begs two questions: how many particles will be emitted; and what controls where they are emitted?

3.1.2 How many particles?

Clearly, you need to be able to control how many particles are created in the scene. There are several ways to do this, but the default method is the **'Rate'** setting. With this selected, the emitter will emit the specified number of particles *per second*.

Since the animation occurs in frames, the number of particles per second has to be divided up into so many particles per frame. For example, with the default rate of 1000 particles per second, if the frame rate is 25 frames per second (fps) then 40 particles are created each frame. After 25 frames, you will have 1000 particles. But note that if the frame rate is 30 fps, after one second you will still have 1000 particles, but you won't see the same number of particles emitted each frame. This is because the number of particles created each frame would be $1000/30 = 33.33...$ recurring. Since you can only have whole particles emitted, 33 will be emitted in most frames, with 34 emitted in one in three frames. This ensures that after one second you have 1000 particles.

So the emitter will act to maintain the rate per second but this can mean varying the number per frame. This won't normally be a problem, but if you absolutely must have the same number emitted per frame, then you can change the birthrate settings. To do this, click the little arrow next to **'Birthrate'** and this will unfold some addition controls:

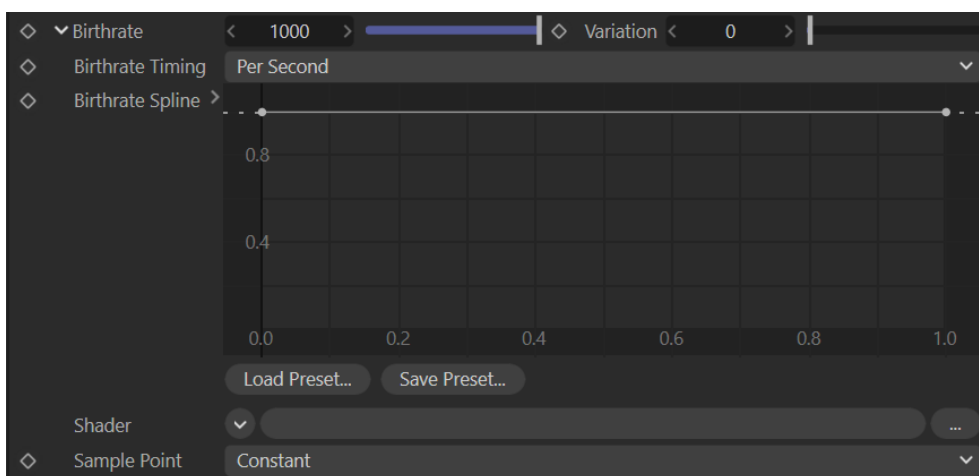
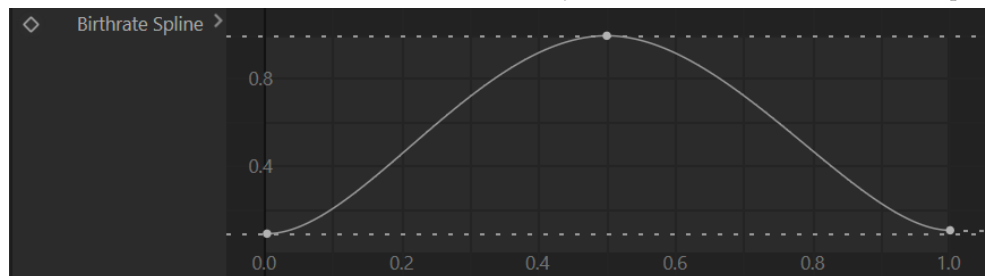


Figure 3.1. Additional birthrate controls

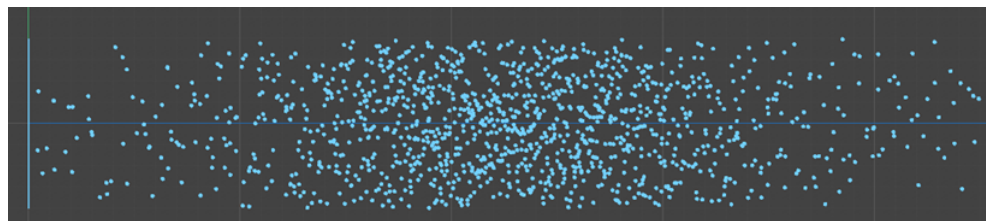
You can change the **'Birthrate Timing'** menu to **'Per Frame'** and now the exact number of particles will be emitted per frame. Do bear in mind that with this setting, if you change the project frame rate in C4D's project settings, the number of particles per frame will stay the same but the number emitted per second will change.

These additional controls enable some further adjustments to the birthrate. For example, you could set the **'Birthrate Spline'** to



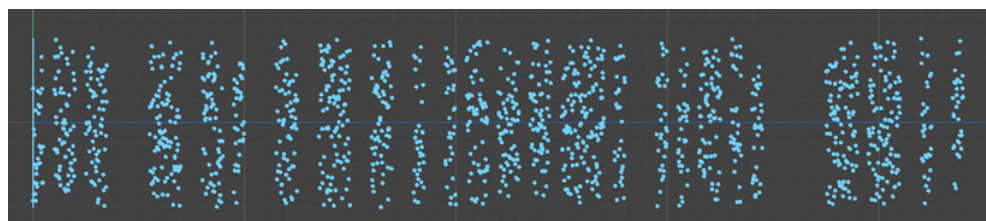
this:

Figure 3.2. Using the Birthrate Spline



Which would taper the emission over the scene length like so:

Figure 3.3. Effect of changing the birthrate spline



Or, you can add a shader or bitmap to the **'Shader'** link. The shader will be sampled each frame and the result used to alter the birthrate. In this image a Checkerboard shader was used with its default settings and in the emitter **'Sample Point'** set to **'Random'**:

Figure 3.4. Using a shader to control the birthrate

3.1.3 Where will they be emitted?

The simplest way is to let the emitter do the work. By default, particles will be created within the boundaries of the emitter. For the default rectangle emitter and others which lie on a single plane, the particles are emitted within the boundaries of emitter and will lie on the same plane. For 3D emitter shapes such as the sphere or box, the particles will be generated at a random position inside the volume enclosed by that shape.

Some of the real power of the emitter shows when you emit particles from a scene object. You can emit from an object's polygons, points, or edges; you can control the emission with a shader or texture tag, or a vertex map; you can constrain emission according to an object's topology, that is, the height and slope of an object's geometry; or you can emit particles only from illuminated areas of the object. How to do some of these things is discussed later on in this chapter.

3.1.4 What else does the emitter do?

The emitter does a lot more than just create particles. It is the emitter which:

- moves the particles to a new position each frame using their speed and direction
- handles collisions between particles and scene objects
- updates other parameters such as colour gradients or rotation (depending on the selected options and the presence of certain

- modifiers in the scene)
- removes particles older than their assigned lifespan or which have been marked for deletion by other XP objects and functions
- calls any particle modifiers in the scene to act on the emitter's particles
- tests any questions in the Questions tab and depending on the results triggers actions which act on the particle

As you see, the emitter not only creates particles but plays a vital role in managing them afterwards. Note that if you remove an emitter from the scene, all its particles are deleted long with it; the particles cannot exist without 'their' emitter.

3.2 Creating particles

This couldn't be easier; add an emitter to the scene, press play on the C4D timeline, and particles are created with their default settings. By itself this won't be that useful, but there's a lot more that can be done. In fact, the emitter has so many settings it can be difficult to know which one to use.

Let's start from the top and look at one of the most fundamental points we need to know: how many particles are we going to get?

3.2.1 Emission

The **'Emission'** setting in the emitter's **'Emission'** tab controls how many particles will be created. There are several options:

Rate

The simplest mode of all, with this setting the emitter emits particles each frame using the **'Birthrate'** setting (and its **'Variation'** parameter) to control how many are emitted. As explained in section 3.1.2 above you may not get the precise number of particles each frame that you might think you will. See that section for details of why this is so and how to make changes to the birthrate.

You can specify whether the emitter should emit particles on every frame in the scene or only between a specified start and end time.

Shot

This mode simply emits a single shot of particles. You can specify how many (note that unlike **'Rate'** this is the number emitted each frame, not each second, unless you disable the **'Per Frame'** switch). Again, you can specify the frame on which to start emission and how many frames it will emit for.

Shot mode also lets you use the hexagonal and regular emission modes, but we'll cover those when looking at fluids.

Pulse

You can think of this mode as being the same as shot mode, except that it emits a shot of particles on a recurring basis, not just once. You can specify how long the pulse (that is, each shot of particles) should be and the interval between the shots. There is one additional control that deserves a closer look - the **'Pulse Spline'** control.

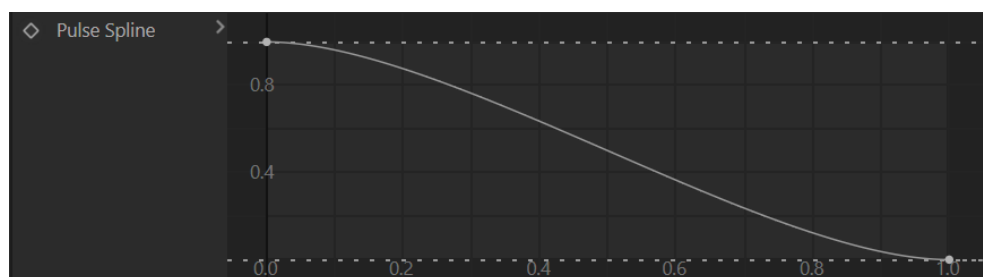


Figure 3.5. Using a spline to control Pulse emission

This spline lets you control the number of particles emitted over the duration of each pulse. For example, you could set it to emit most particles on the first frame of the pulse and fewer on the last frames to give a kind of 'feathered' appearance to the trailing edge of the pulse. This is shown in the example on the next page. The left side of the image shows the default spline (which has no effect) and the right side shows the feathered edge.

Figure 3.6 shows the effect which results.

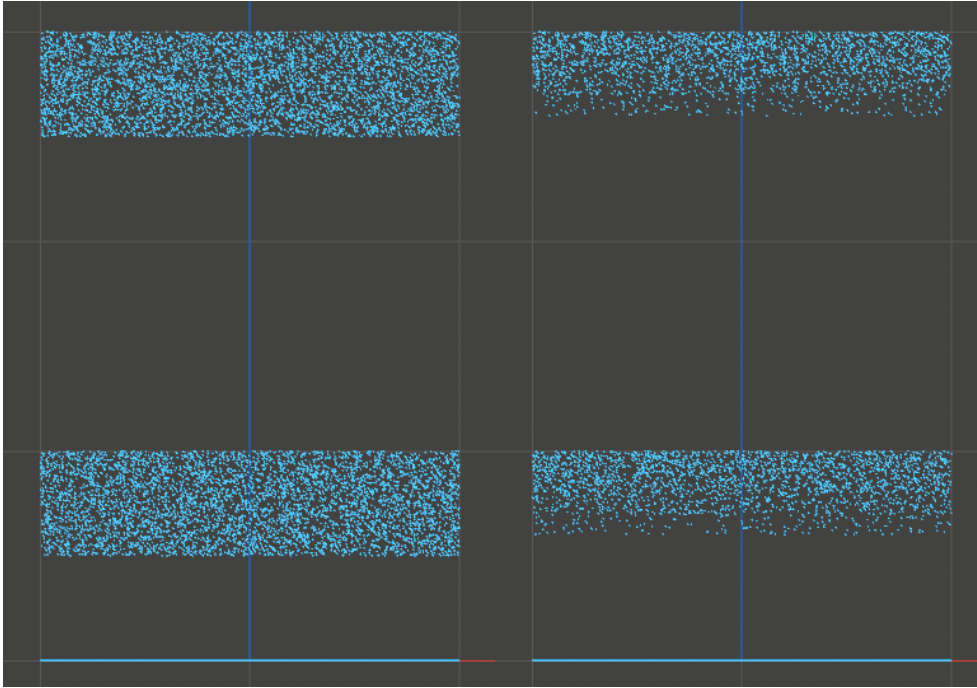


Figure 3.6. Effect of using the Pulse Spline to produce a 'feathered' edge to the emission

To see the best results from the spline the pulse duration should be several frames long, or you will see little or no effect.

Regular and Hexagonal

We'll leave these for the time being as they are optimised for fluid simulations so will be covered in that section.

Trigger

This may be the least understood, and certainly the least used, of the various emitter modes, so it is worth looking at it in a little more detail.

The problem with the modes discussed so far is that you cannot set the emitter to emit particles only when instructed to do so, such as if a particular condition is fulfilled. This is what 'trigger' mode is intended for. When you first enable this mode, it behaves in exactly the same way as **'Rate'** mode. But if you disable the **'Pull Trigger'** switch, no particles are emitted; checking this switch again, even during playback, will immediately cause the emitter to start emitting particles. This isn't that helpful; after all, you could simply set a starting frame for the emitter in **'Rate'** mode and get the same result. The real power comes when you use some other condition to start emission.

In this scene, there are two emitters. The orange one is set to pulse emission, emitting one particle every 20 frames. These particles collide with the cube object, and each time there is a collision the **'Change Emitter Action'** is triggered. This 'pulls the trigger' on the green emitter, causing it to emit a burst of particles. To do that, the green emitter must be set to trigger mode and the **'Trigger Count'** set to **'Set by Action'**. The result is shown in Figure 3.7 on the following page.

🗣️ See the [full animation](#) on the book's website. The scene file [ch3_emitter_trigger.c4d](#) can be downloaded as part of this chapter's example files archive.

Controlled Only

Finally, there is the mode called **'Controlled Only'**. You won't usually want to set that yourself. It tells the emitter not to emit anything until told to do so by some other object. This was originally used by the Spawn modifier, but there are a number of other objects which use it, such as the Dynamic Particles modifier, Foam object, Collider tag and so on. It enables the objects which use it to generate particles from an emitter only when required, often with additional parameters that they set. This option is available not because you might want to set it, but because you might need to turn it off.

Suppose you add a Spawn modifier and drag an emitter into its **'Spawning Emitter'** link field. Doing this will automatically set the emitter mode to **'Controlled Only'** and you will see this because the emitter icon will change in the object manager. Now what happens if you delete the Spawn modifier? Unfortunately Cinema 4D does not inform linked objects that the object they were linked to has been deleted, so now you have an emitter which does nothing at all. To fix this, simply change the emission mode to some other mode and it will work (and the emitter icon will change back to normal).

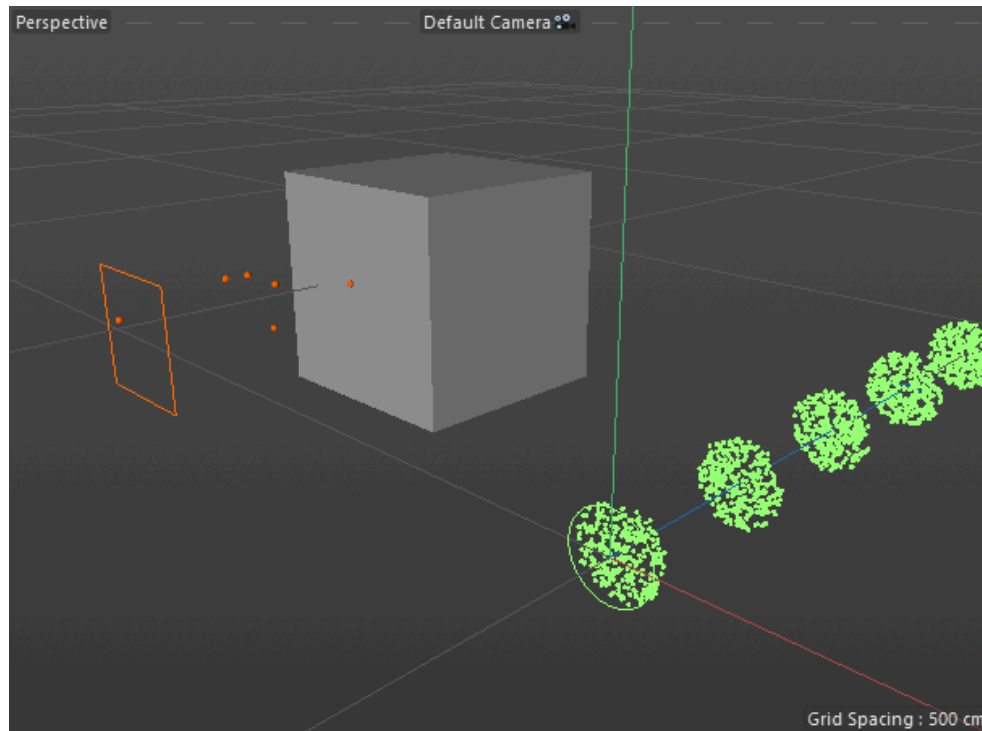


Figure 3.7. Using Trigger mode to trigger emission on particle collision

3.2.2 Subframe emission

When particles are emitted, the emitter creates the particle then moves it to a new position using the velocity assigned to it. If all particles have the same velocity they all move the same distance. The next frame the existing particles are moved and a new set created and moved as before. The result is that the particles appear to be emitted in bands or stripes; this can be very obvious if they are all moving in the same direction and with the same speed. Subframe emission prevents this. You can see the effect in Figure 3.8. For the first 30 frames, subframe emission was disabled; you can see the result in the blue particles. For the next 30 frames (orange particles) it was enabled, and you can see that the banding effect has gone:

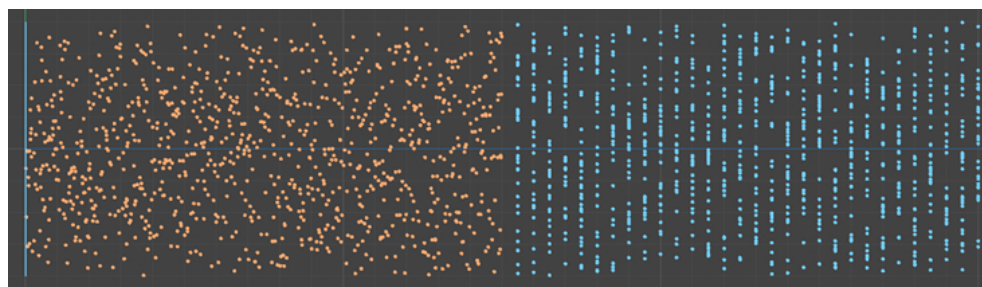


Figure 3.8. Effect of subframe emission to reduce banding

3.2.3 Spawning particles

There is another way to create particles and that is to 'spawn' them from an existing particle. Several objects in XP enable this, and the simplest example is the Spawn modifier. This modifier will generate additional particles from a 'source' particle, using a separate emitter.

The way the modifier works is very simple. For each particle which is affected by the modifier (that is, which is in a particle group which the modifier works on, from an emitter which the modifier works in, and is within the field of effect of the modifier) additional particles will be generated from the separate, or spawning, emitter.

A simple example will demonstrate this. In this scene the Spawn modifier affects only particles from the 'Main emitter'. A separate emitter, 'Spawning emitter' will only emit particles when instructed to do so by the Spawn modifier. The modifier itself has a spherical field, so only when the blue particles from 'Main emitter' are in the field are the orange particles spawned. Once they leave the field, no new particles are created from them:

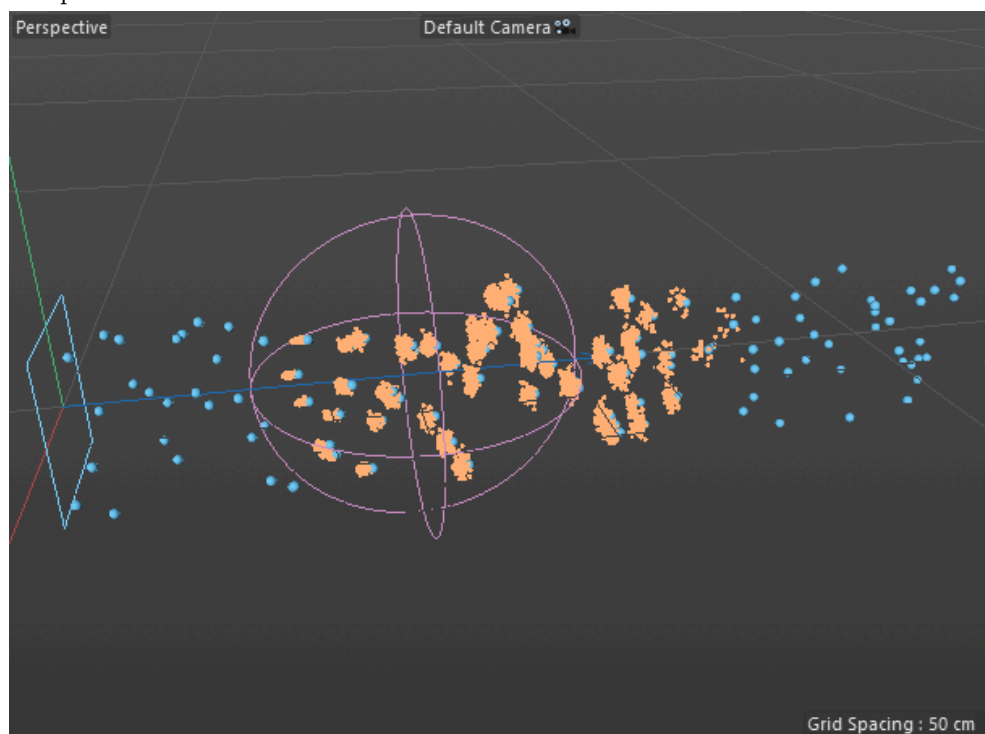


Figure 3.9. Particle spawning

See the [full animation](#) on the book's website. The scene file [cb3_emitter_spawning.c4d](#) can be downloaded as part of the book's example files archive.

As well as the Spawn modifier, there is also a Spawn action, and some other XP objects also include the ability to spawn particles. It's a very useful technique for all sorts of effects such as trails of particles, fireworks, dust on collision, etc.

3.2.4 Simulation modes

The emitter has three simulation modes which you access with the **'Mode'** menu in the **'Emissions'** tab. The default is **'Simulate (Legacy)'**, so-called because it was the first, and for a long time, only mode available. The newer version of this is **'Simulate'**.

There is only one difference between these modes, but it's an important one. In the legacy mode, no particles are emitted on the start frame of the animation, normally frame zero. That's because rewinding to the start frame was used to clean up all the data used by the emitter, free memory, and so on, giving a clean slate for the animation to start again when playback was commenced. The problem here was that in some cases a scene required that particles were present in the scene right from the start. To enable this **'Simulate'** mode was added.

You can use whichever of these modes you prefer, depending on whether you want to start with a scene empty of particles or for some to exist right from the start.

'Motion' mode is very different. Selecting this mode causes particles to be generated immediately as with **'Simulate'** mode, but now in each frame the emitter moves the particles to where they would be after 1 second's playback time. Not only that, but every particle has a lifespan of one frame, after which it is killed and recreated. Since each particle is recreated using the same parameters as the previous ones, they all appear to be fixed in the same place and not moving at all.

You might ask what the use of this is. The XP manual shows one very useful effect: if you are generating particles by means of an object's texture you can animate the texture and the particles appear to move with it. In fact, they aren't moving, they are being recreated each frame. Figure 3.10 is a still from another example of this technique.

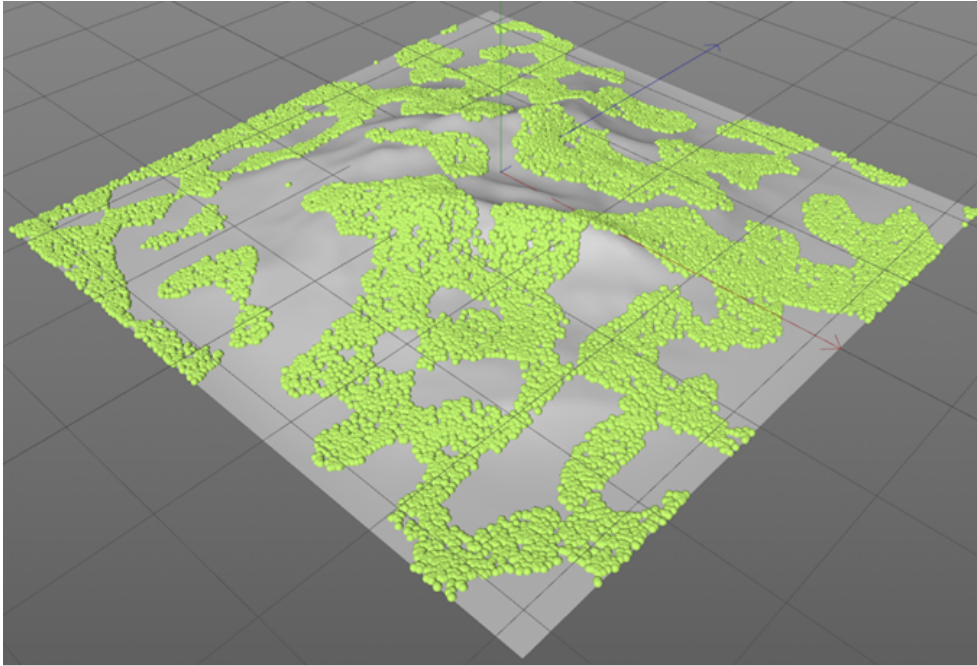




Figure 3.10. Emitter 'Motion' mode

 You can see the [full animation](#) on the book's website. The scene file [cb3_emitter_motion_mode.c4d](#) is included in the download archive for this chapter.

3.2.5 Motion inheritance

This seems to be a little-used feature of the emitter but can be quite useful. Suppose you have an emitter which oscillates on the Y-axis, moving up and down in the scene (e.g. by using a Vibrate tag). The fact that the emitter moves up or down has no effect on the particle direction. It does affect the position at which particles are created, but they will move with the direction given to them by the emitter, which is along the positive Z-axis by default.

What motion inheritance does is set the initial direction of newly-created particles to match that of the emitter combined with the default particle direction. It doesn't affect the particle movement once it has been emitted, it simply alters the direction on emission to blend the default particle direction with the direction of the emitter movement/rotation.

 You can see an example in [this animation](#) on the book's website. In this animation, the emitter has a Vibrate tag attached to it and motion inheritance is enabled. For the first part, the Vibrate tag is disabled and the particles behave as expected from a spherical emitter (these are the blue particles). At frame 90, the tag is enabled (and the particle colour is changed to make the effect easier to see). Figure 3.11 is a still from the result at about frame 140.

The scene file [cb3_emitter_motion_inheritance.c4d](#) is part of the chapter's example files archive if you want to see how it's done.

3.2.6 Generating Thinking Particles

One of the main reasons for developing X-Particles was to remove the need to use the antiquated, Xpresso-driven Thinking Particles, which hasn't been updated in C4D for (literally) decades. So it may seem a little odd to still want to do this. It was included in XP so that users could use effects such as the Pyrocluster shader or the Mograph Tracer, both of which required Thinking Particles. More recent versions of XP remove the need to do this thanks to Explosia FX and other features which remove any requirements for Thinking Particles, but they are still included in case users have older scenes that use them.

This won't be covered further here. If you really need to generate Thinking Particles, the XP manual has all the explanation required.

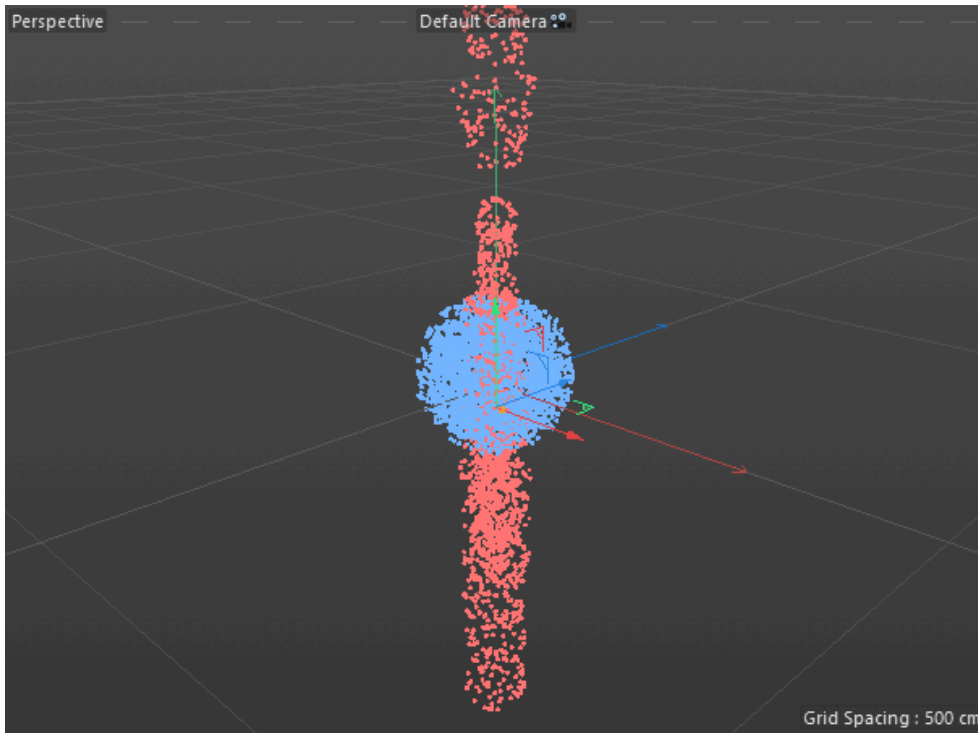


Figure 3.11. Motion Inheritance

3.2.7 Setting a maximum number of particles

You may have noticed that there is no way to set a maximum number of particles to be emitted. Of course, you can use shot or pulse or trigger mode to emit a fixed number but what if some are destroyed during the animation and you need to emit more to make it back up again? Or you want to use rate mode to emit every frame until you've emitted a specific number of particles?

In fact, you can't do this in the current version of X-Particles. That functionality used to be included but was removed for technical reasons. For this reason I've written a small add-on plugin which will let you set the maximum number of particles in the scene. You can download it as part of the add-on package for this book; full documentation is included with it. The plugin can be [downloaded here](#).

3.2.8 Initial state

This is actually found on the **'Object'** quicktab of the emitter but it is most appropriately discussed here.

Suppose you have emitted some particles and they are just how you want them, but you need them to be like that at the start of the animation. You could, of course, alter the timeline so that emission took place before frame zero, then only render from frame zero, when the particles have already been generated. But a much easier way is simply to get the particles how you want them and then set that to be the initial state of the emitter. If you use the **'Simulate'** emission mode, those particles will then exist from the start of the animation.

To do this is very simple. Get the particles how you want them, then in the **'Initial State'** quicktab, click the **'Set State'** button. If you rewind the animation to the start, the emitter will be reset so that rather than clearing all the particles, it will automatically generate them to match the initial state. If the emission mode is **'Simulate'** then they will exist from the first frame; if it is 'Simulate (Legacy)' you will need to advance one frame to regenerate them.

You can temporarily disable the state by disabling the **'Use Initial State'** switch. This doesn't delete the saved state, so you can go back to it again by enabling the switch. To delete the saved state altogether, click the **'Clear State'** button.

One interesting thing you can do is save and load the initial state. To save the state, click the **'Save State...'** button, give it a filename (no extension is needed - the emitter will use the same extension as X-Particles cache files) and the state is saved to disk in the same format used by the XP Cache object. With this you can save multiple states and choose the one you like best. To load an initial state, just click the **'Load State...'** button and select the file you want.

Using the Cache object with initial state

Since the Cache object and the initial state files use the same format, you can cache a scene and then load one of the cache files into the initial state of another emitter, even in a different scene file. To do this, you would:

- cache the scene using external files (you can cache the whole scene or just one or more frames if you don't need it all)
- create another emitter, which could be in the same or a different scene file
- load the initial state for that emitter, selecting one of the cache files to load
- play the animation, which will now start from the selected cached frame from the other emitter

3.2.9 Painting particles

There is one other way to create particles, which is to paint them on the surface of a polygon object. This can be very effective but is a little...quirky to use, so will be covered at a later point in this book.

3.3 Different emitter shapes

The emitter, by default, is a rectangle. Particles are emitted from the area enclosed by the rectangle and travel in a set direction, which is along the positive Z-axis of the emitter. The size of the square can be changed, you can choose to emit only from the rectangle edges, and so on.

However, you aren't restricted to one shape. Rectangle, Circle and Ellipse emitters emit particles only on one 2D plane, but Sphere, Box, Cylinder and Torus emitters all have 'volume' so are useful if you need particles to occupy a volume of 3D space.

These are all useful shapes but the real power to control emission comes when you emit from an object. This is discussed in detail below.

3.4 Defined emission

In Cinema 4D the Mograph Cloner can be thought of as a (relatively limited) particle engine. The purpose of defined emission is to add some of the capabilities of the Cloner to X-Particles.

What it does is enable the emission of a fixed number of particles at a fixed position. If you set the particle speed to zero and the emission mode to **'Simulate'** and link a Generator or Sprite object to the emitter you have in effect a simple cloner. You can even use Mograph effectors on the emitter but - and this is the real advantage over the Cloner - you have all the power of the XP engine to manipulate the particles and any linked objects.

3.4.1 Using defined emission

This is straightforward. Switch to this mode in the Object tab of the emitter and select what generation pattern you would like - a single line, a circle or a 3D grid. Each small box is where a particle will be emitted and you can change the number, the spacing between them, and so on.

The only unusual option is **'Fragmenter List'**. This is for use with the XP Fragmenter object and this option will be covered when we look at that object.

You can set the initial particle direction as usual but note that the circular mode gives you an extra option. If you enable the **'Radial'** switch the particles will move in a direction radially from the centre of the circle.

3.4.2 Other controls

To simulate the behaviour of a cloner, the emitter only needs to emit one particle per box in the viewport. You could set this up easily enough in the Emission tab of the emitter, but as a shortcut you could simply enable the **'Static Mode'** switch. This will force the emitter to emit just one particle at the location of each box. Most of the time you would want this to happen on the first frame of the animation, but you can choose the emission frame by changing the frame number in the **'On Frame'** setting.

3.5 Instances

One problem with the emitter is that you cannot clone it in a Mograph Cloner. If you try, only the original emitter will emit any particles. To get around this you can use an 'Instance' emitter.

An instance emitter takes all its properties from a source emitter, including shape, birthrate, particle speed, colour, and so on. To make an instance emitter, in the 'Object' tab change 'Emitter Shape' to 'Instance'. Then drag and drop another emitter, which will be the source emitter, into the 'Emitter' link field. You can then edit the source emitter as required and the instance emitter will duplicate the parameters.

The interesting part comes when you make the instance emitter a child of a Cloner. Then you get multiple copies of the same emitter. One small issue is that the cloned emitter shape does not show in the viewport, so you can't see where the clones are until you start emitting particles. To get round this it's helpful to make some other object a child of the instance emitter so you can see where the clone is located. In this image, the source emitter is the circle on the left of the screenshot and a circle spline of the same size has been made a child of the instance emitter so you can see where the clones are:

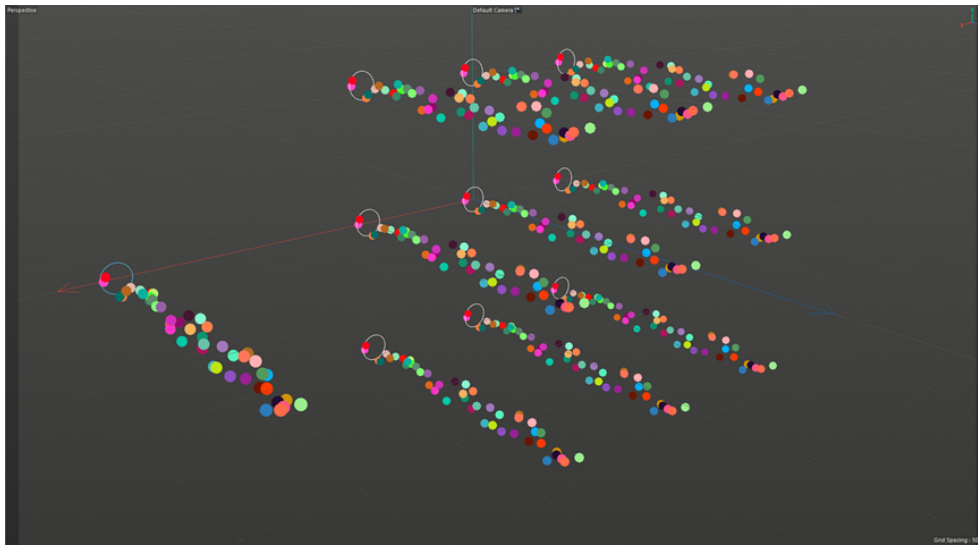


Figure 3.12. Using an Instance emitter in a Mograph cloner

The scene file is [ch3_emitter_instances.c4d](#) and is present in the download archive for this chapter.

⚠ You should be aware that the instance emitter is an exact copy of the source emitter in almost all respects. If the source emitter particles change colour, so will the instance emitter particles. If the source particles collide with an object, the instance particles will behave as if they have collided too, and note that the instance particles will not collide independently with any scene object, only the source particles will do so.

You might wonder in the screenshot above how to stop the source emitter from emitting particles if all you need is the clones. In short, you cannot do so - if the source emitter doesn't emit anything, the instance emitter won't do so either. You can either move the source right out of the scene so it doesn't interfere, or if you just don't want to see the source particles you can disable the 'Show Particles' switch in the source emitter's Display tab. But remember, those particles are still there and will interact with the scene with regard to collisions and so on.

The 'Show Particles' switch is one of the parameters not taken from the source emitter. There are other such parameters - the source and instance emitters can both show their own HUD, for example.

Summary

At this point we've covered emission of particles from a simple emitter. The next chapter looks at something a little more complex and a lot more interesting: emitting particles from an object in the scene.

Chapter 4: The Emitter: Emitting from an Object

In this chapter we will look at how to emit particles from an object and the many ways in which to control the emission.

↓ The scene file(s) for this chapter can be found in the archive [examples_ch4.zip](#) on the book's website. Click here to [download the archive](#).

4.1 Changing to object emission

As we have seen, you can emit particles from a basic emitter which can have one of several possible shapes. But if you emit from a scene object, you have complete control over where the emission takes place.

To do this, in the **'Object'** tab of the emitter, change **'Emitter Shape'** to **'Object'**. This will change the emitter's interface and you will have all the options for object emission.

4.1.1 Providing an object

The first thing, of course, is to feed the emitter with an object to emit from. Without that, nothing will be emitted. You can use any polygon or spline object, including primitives or editable objects. You can also use generators such as the Cloner or Array objects or even those generated by another XP object. Drag the object into the **'Object'** link field and the emitter is ready to emit particles.

i You can't emit from a Null object, but you can make one or more other objects to be child objects of a Null, then drop the Null into the **'Object'** link field. Particles will then be emitted from the child objects of the Null. Also see the **'Object Chain'** parameter in the 'Other options' section.

4.1.2 Selections

You aren't restricted to emitting from the entire object. You can create a selection of polygons, points or edges on an object and drag the selection tag into the **'Selection'** link field. What you actually get will depend on the nature of the selection and the object, and where you want to emit from.

For example, if you have a polygon selection and want to emit from polygons, this works as expected. If you emit from points, particles are emitted from all the points making up the polygons in the selection; but if you emit from edges, no particles are emitted. This is discussed further in the 'Where to emit from' section below.

Certain Cinema 4D objects - such as Extrude, Loft, Lathe and Sweep - provide hidden selections which are for the caps and bevels on the object. Some later versions of C4D can generate selection tags for these areas even if the object is not yet editable; earlier versions cannot do so. However, regardless of C4D version, you can specify whether to emit only from the object body, or its caps, or both, using the **'C4D Generators'** menu setting. If you select **'Use Body and Caps'** or **'Use Caps Only'** you can expand the options below this menu by clicking the little arrow next to the menu and select which of the four selection tags you want to use. By default, all are selected but you can turn off emission from one of the caps, for example, by using these switches.

! Note that if you are using a version of C4D which does produce selection tags on the parametric object, you cannot drag these into the **'Selection'** link field - they won't work. You have to use the menu and switches described above.

4.1.3 Vertex maps

The **'Selection'** link field can also accept a vertex map. If you emit from the object's points, emission will only take place from a vertex if the vertex weight is greater than zero. You can also increase the threshold from zero using the **'Threshold (Vertex Map)'** setting.

4.2 Where to emit from

Where on an object should the emitter emit particles from? X-Particles provides a large number of possibilities for this, which will let you control the site of emission very closely. Some are very simple while others are more complex.

4.2.1 Polygons

XP has four ways to emit from polygons without any further controls. If the object has no polygons, either no particles are emitted or in the case of a spline they are emitted from anywhere along the length of the spline.

Polygon Centre

The simplest method and the default setting. Particles are emitted from the centre of a polygon. If an object has any n-gons, these are converted internally into ordinary polygons before emission.

It might be of interest to look at how XP goes about this. Essentially, the algorithm works like this:

1. Each frame, calculate the number of particles to emit
2. Randomly select a polygon from all the polygons in the object or a polygon selection if provided
3. Find the centre of that polygon
4. Emit a particle from that position

There's a little more to it than that, but you can see that if the object has more polygons than the number of particles to be emitted in a frame, not all polygons will emit a particle. It is also possible that some polygons may emit more than one if they are randomly selected more than once.

Polygon Area

This is more interesting because particles are emitted anywhere on the surface of a polygon, not just its centre. The algorithm is the same as for polygon centre, but step 3 above would read 'find a position on the surface of that polygon'.

There is a problem, however. Suppose you have an object with polygons of very different sizes - some very large, some very small. Randomly selecting a polygon means that each polygon has an equal chance of being selected, which in turn results in the same number of particles being emitted on average from small polygons as large ones. This can result in the appearance of particles being unevenly distributed over a surface.

To fix this, '**Polygon Area**' applies a weighting to its random selection of a polygon which is calculated from the size of the polygon relative to all the other polygons in the object. This means that large polygons are more likely to be selected than small ones, so large polygons emit more particles and they appear more evenly distributed over a surface.

Polygons

This is the same as polygon area, but with one important difference: there is no weighting for polygon size, so each polygon has an equal chance of emitting particles, regardless of its size. Whether you choose this algorithm or polygon area depends entirely on what you are trying to do.

Here you can see the difference between '**Polygon Area**' and '**Polygons**':

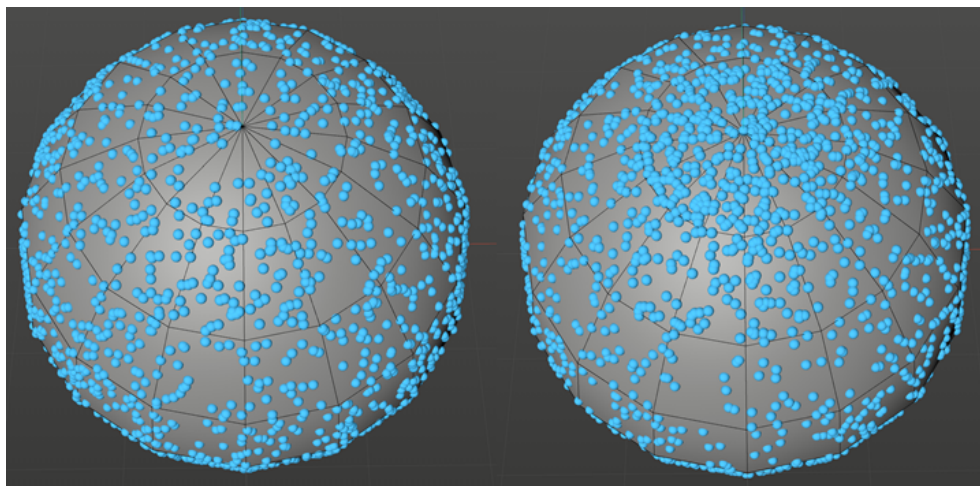


Figure 4.1. 'Polygon Area' vs. 'Polygons' mode

'Polygon Area' is on the left. Note how on the right **'Polygons'** causes the smaller polygons at the pole of the sphere to emit more particles and that larger polygons emit fewer, so the distribution looks uneven.

N-gon Centre

This is a special mode in which particles are only emitted from the centre of an n-gon. If the object has no n-gons, no particles are emitted.

4.2.2 Points

This will cause the emitter to emit particles from the object's vertices or a point selection. The only other option you have in this mode is to colour the particles using the vertex colour. This requires a Vertex Color tag, which is only available in C4D R18 and higher.

4.2.3 Edges

This is a complex mode because the options depend on the type of object, either polygon or spline.

Polygon edges

In this case, the particles are emitted from the edge between two of the points which are part of the polygon. There must be an actual edge, that is, the two points must be part of the same polygon. The emission position along the edge is chosen randomly, unless you turn on **'Even Distribution'**. Then the particles are spaced equally along the edge; the gap between them is controlled by the **'Spacing'** parameter, with lower values leading to smaller gaps.

Splines

There are a number of additional options when using splines and edge emission. Particles are emitted from the portion of the spline between two adjacent points on the spline. If you set **'Emission Point'** to **'Random'** there are no other controls available and the entire spline will be used (point selections are ignored).

However, if **'Emission Point'** is changed to **'Set'** you have several options. The first is **'Position'**. This is the starting point for emission along the spline; the default is zero, that is, the start of the spline. If you set it to 25%, emission will start at a quarter of the way along the spline. Although you can set this to negative values, that doesn't really offer anything different, because -75% along a spline is the same position as plus 25% along it. Likewise, values over 100% have no additional meaning (125% along the spline is the same as 25% along it).

The second and very important control is **'Spread'**. This controls how far along the spline from the **'Position'** setting particles will be emitted. By default this is an even spread in both directions from the **'Position'** point. For example, if you set **'Position'** to 25% and **'Spread'** to 40%, particles will be emitted from 5% along the spline to 45% along it. This is because the spread of 40% takes place either side of the start point. In this case, the start point is 25% so emission will extend 20% of the length back from that point and 20% forward from it. ($25\% - 20\% = 5\%$; $25\% + 20\% = 45\%$.)

You can force the spread to go forward only by enabling the **'Unidirectional'** switch. With the above example, turning on this switch will cause emission to take place from 25% along the spline to 65% along it ($25\% + 40\% = 65\%$).

You can see that depending on the values of these two settings, particle emissions can wrap around the spline. For example, if **'Position'** is 80% and **'Spread'** is 30%, with **'Unidirectional'** enabled, emission will take place from 80% to 110%, which will cause particles to be emitted from the start of the spline. This might not matter to you, especially with closed splines, but if the spline is open it can look rather odd. To prevent this from happening, which might be difficult if you keyframe these values, simply enable the **'Do Not Wrap Emission'** switch.

If you are emitting from a multi-segment spline, the emitter will by default emit from all the segments. You can alter this behaviour by setting **'Segments'** to **'Select'** and choosing the segment the emitter will use. Particles will not be emitted from other segments. Note that the emitter does not check that you have entered a valid segment number; if the number is invalid (in other words, if there is no such segment) no particles will be emitted.

Finally, as with polygons you can turn on even distribution and set the spacing. However, very importantly, the **'Spacing'** parameter for splines works in the opposite way to polygons; for smaller gaps you need to increase the value, not decrease it.

4.2.4 Texture-controlled emission

This is one of the most interesting emission modes. With it, you can control emission based on the texture applied to an object, or by a shader.

When you select **'Texture'** from the **'Emit From'** menu a new quicktab appears in the interface labelled **'Texture'**. All the settings for texture emission are in this quicktab. From the start you have an important choice to make. Will the emission be controlled by a texture, using a texture tag applied to an object, or will it be controlled by a shader without a tag?

The more powerful option is to use a texture tag. If you use a shader, you can add it to the **'Shader'** link field. The shader will not affect any material on the object, so this can be useful if you don't want to change the object's material. In this example, the sphere object has a Checkerboard texture applied but the emission is controlled by a noise shader in the **'Shader'** field and is not affected by the texture (and does not affect the texture in any way):

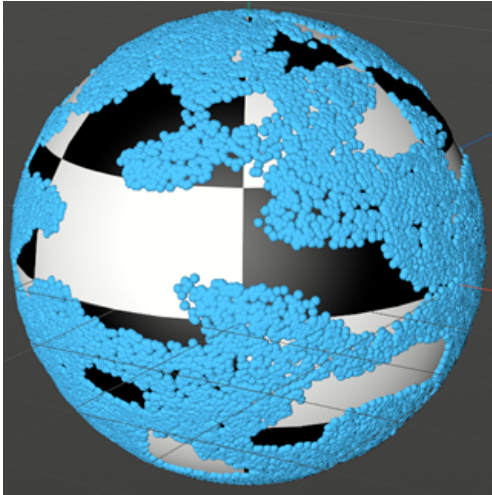


Figure 4.2. Emission controlled by a shader

Apart from any changes to the shader itself, this mode gives you little control over the particles. You can opt to have the shader colour affect the particle speed or colour or neither, as shown above, but that's about it. For greater control, use a texture tag instead.

If the texture tag from the sphere is dropped into the **'Texture Tag'** link field, the result looks like this:

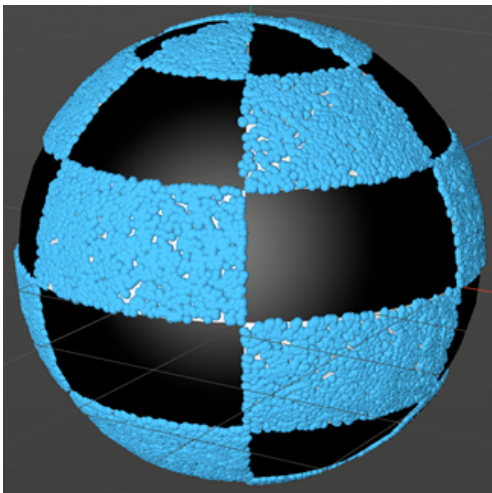



Figure 4.3. Emission controlled by texture tag

The particles are emitted from the white areas of the texture but not from the black ones (this can be changed with some additional settings in this quicktab - see below).

⚠ The emitter can cope with almost all the projection modes in the texture tag with the exception of frontal and camera mapping.

What happens if you have both a texture tag and a shader? In this case the tag takes over completely and the shader is ignored. There is no blending or anything similar between the two.

 The texture tag doesn't have to come from the object used as the emission source - it can be a tag from another object altogether if desired.

Channel options

Below the tag link field there are several menus labelled '**Emit Channel**', '**Color Channel**', etc. These define which channels in the material affect which particle parameter. By default, the texture only affects emission and particle colour; all the other menus are set to '**None**' which means that they will have no effect.


Using these channels is simple. Suppose you want the particle speed to be changed by the texture. In the '**Speed Channel**' menu, set the channel to something other than '**None**'. The colour channel will be often be used, but it doesn't have to be. You can use any of the channels in the menu, even if that channel isn't being used in the material itself. In other words, the material might not use transparency at all, but there's no reason you can't use the transparency channel in the material to control the particle speed. You simply have to ensure that there is something in the transparency channel to affect the speed of the particle. If you added a noise shader with a small scale, for example, this would be a good way to get a random particle speed.

The only channel with additional options is the scale channel. With this you can alter the particle scale and the extra options give you more control over what is changed. The manual explains these options so they won't be covered again here.

Other options

These are available for both shader and texture modes and add more controls for particle emission (they don't affect colour, radius and so on).

The first option determines how the texture controls emission. After all, the texture is just a colour - how does that change particle emission? The default value is colour brightness. The brightness of a colour is calculated by taking the average of the red, green and blue RGB channels. This gives zero for black, 1 for white, 0.33 for pure red, and so on.

 Note that pure blue and pure green also have brightness values of 0.33 so would give the same results as pure red. The actual colour doesn't matter: it's the brightness which is important.

This value is then multiplied with the probability of a particle being emitted from a point on the surface. For black areas, since the brightness is zero, no particles will ever be emitted, but they always will be from white areas. For brightness values in between the effect is to see more particles the higher the brightness and fewer if the brightness is low.


This can be inverted using the '**Invert**' switch. With the default settings, this means that particles will be emitted at the highest rate from pure black areas, but none will be emitted from white areas.

One potential problem is that you may have set up a texture as you want it but then find that too many particles are being emitted from darker areas (or conversely that not enough are emitted from brighter areas). You can fix this with the '**Contrast**' slider. The default setting is zero, but if you increase it, fewer particles will be emitted from darker areas, so more will be emitted from brighter areas. If you decrease the value, more particles are emitted from darker areas and fewer from brighter areas.

There is also a '**Mode**' setting. This switches between using the colour brightness to control emission and the proportion of red, green and blue in the sampled texture. For example, if this is set to '**Red**' then only areas which have some red in the texture will emit particles. So a texture which is pure green and pure blue with no red will emit no particles.

In the example file [cb4_emitter_texture_emit.c4d](#) in the download archive the texture is a noise with black and orange; the orange is 100% red with 50% green. Try the various modes and see what result you get.

Animated textures

 It is very simple to emit from an animated texture. In C4D, the Fire shader is automatically animated. Figure 4.4 is a still from the [animation](#) on the book's website shows what it looks like applied to a Plane object and rendered (no particles, this is just the shader):

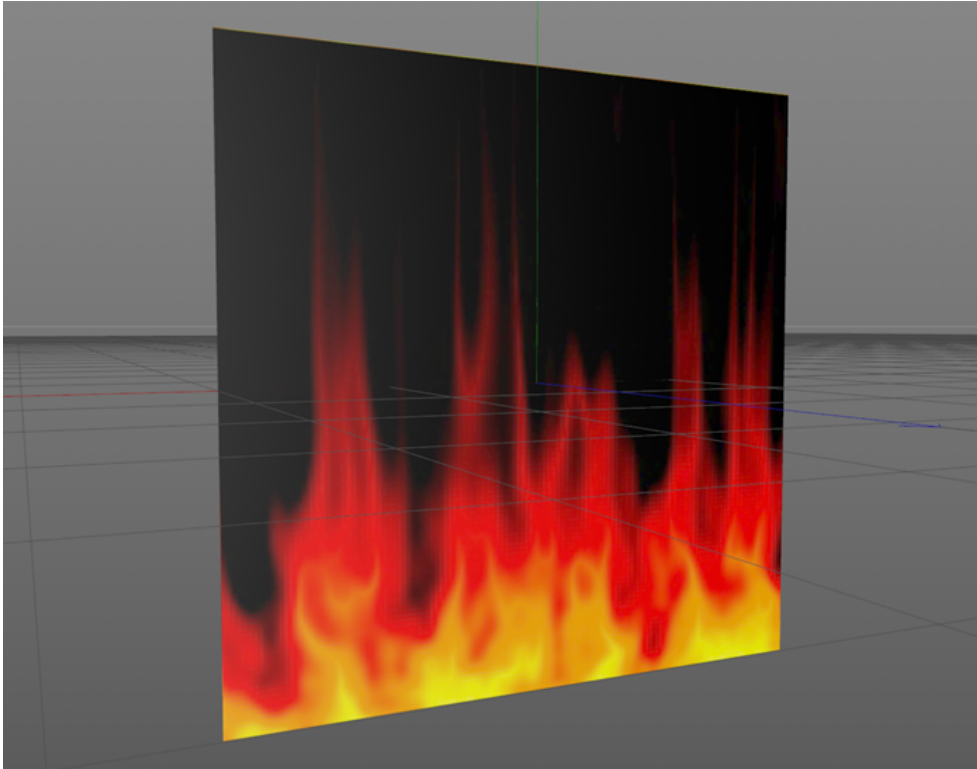


Figure 4.4. Fire shader on Plane object (no particles)

In Figure 4.5 the shader is not rendered at all, what you see are entirely particles generated under the control of the Fire shader in the **'Shader'** link field:

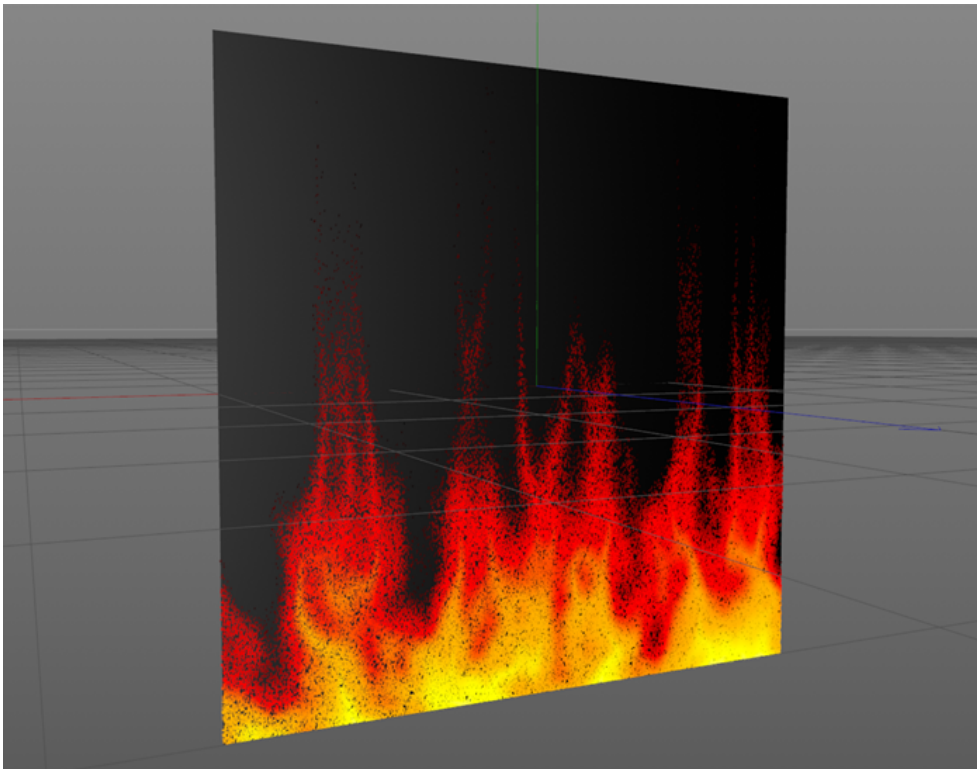


Figure 4.5. Particles emitted from the Plane under shader control (shader not rendered)

 You can also see [this version](#) on the book's website.

The shader version is smoother of course, but to get near that with particles would require a very high particle count. Even so, you can see the animated shader giving the correct colour to the particles. The particles themselves have a very short life (three frames) and

that could be reduced - or you could use the **'Motion'** emission mode.

The scene file [cb4_emitter_anim_texture.c4d](#) is in the file archive for this chapter if you want to see how it works. There are a couple of other options in this section, which are explained in the manual.

4.2.5 Emission from object colour

All polygon objects in C4D can have a colour (set in the Basic tab of the attribute manager). You can use this to control emission. This might not seem very useful. An object can only have one colour, after all. But what if you are producing multiple objects which then have different colours?

In this scene, a Cloner is generating 10 spheres, and a Random effector has been used to vary the colour. On the left you see the actual spheres, on the right the spheres have been hidden and you see particles generated from them. You can see that the particles adopt the colour of their source object and that objects with darker colours emit fewer particles than those with bright colours:

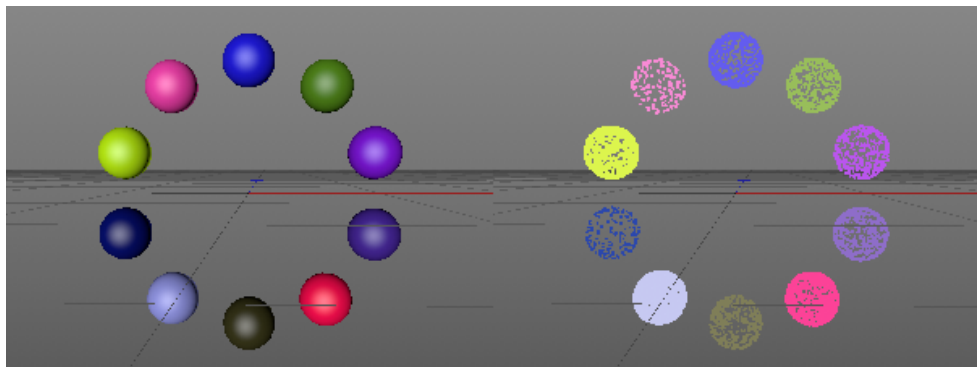


Figure 4.6. Emission from an object's colour

This file [cb4_emitter_objcolor.c4d](#) is in the download archive for this chapter.

⚠ This only works with polygon objects and polygon primitives, not splines.

4.2.6 Illumination-controlled emission

This method controls emission depending on the illumination of a surface by the light. To use it, select **'Illumination'** from the **'Emit From'** menu and the **'Illumination'** quicktab appears. You must specify the light or lights which will control emission by dragging and dropping them into the **'Lights'** list.

⚠ Note that other lighting effects such as global illumination, image-based lighting, the physical sky and so on can't be used - it must be an actual Light object.

In Figure 4.7, the sphere is illuminated by two lights, one blue and one red. If both lights are excluded (a red cross against them in the **'Lights'** list) no particles are emitted. If either or both are included (green tick in the list) particles are emitted when the light(s) illuminate the surface. The **'Inherit Color'** switch has been enabled so the particles are coloured by the colour of the light which illuminated the surface point where they were emitted.

Increasing the **'Brightness Threshold'** will constrain the particle emission to areas which exceed that level of illumination. In Figure 4.8, the value has been set to 70%.

This scene file [cb4_emitter_illum_1.c4d](#) is contained in the download archive for the chapter.

💡 The light can be animated. See [this animation](#) on the book's website. This uses a Vibrate tag to rotate a spotlight; particles are emitted only where the spot illuminates the surface. The scene file [cb4_emitter_illum_2.c4d](#) is also in the download archive.

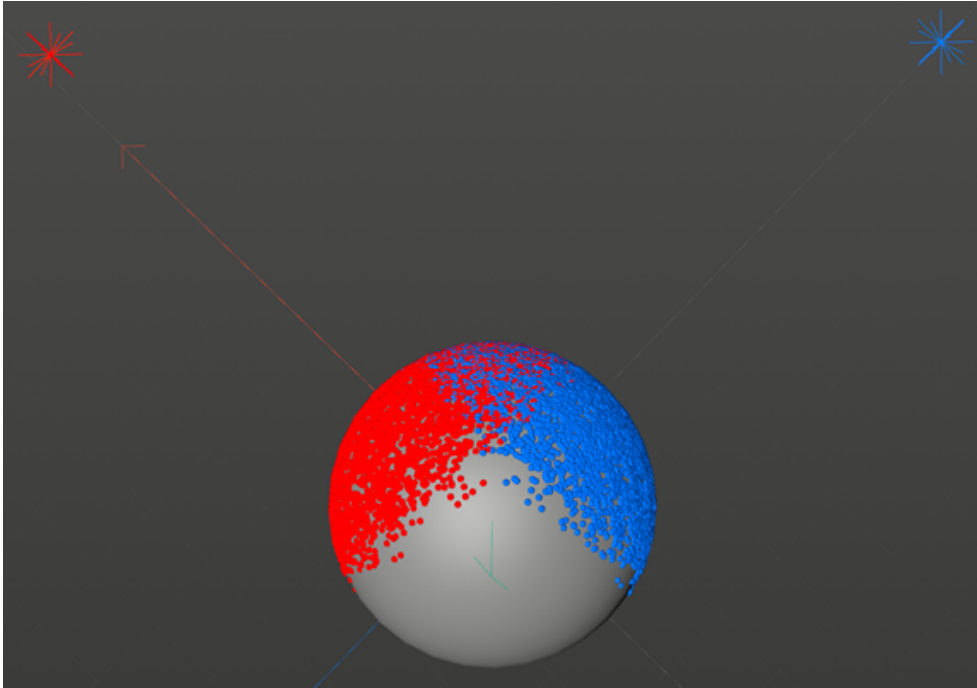


Figure 4.7. Emission controlled by illumination from a light object

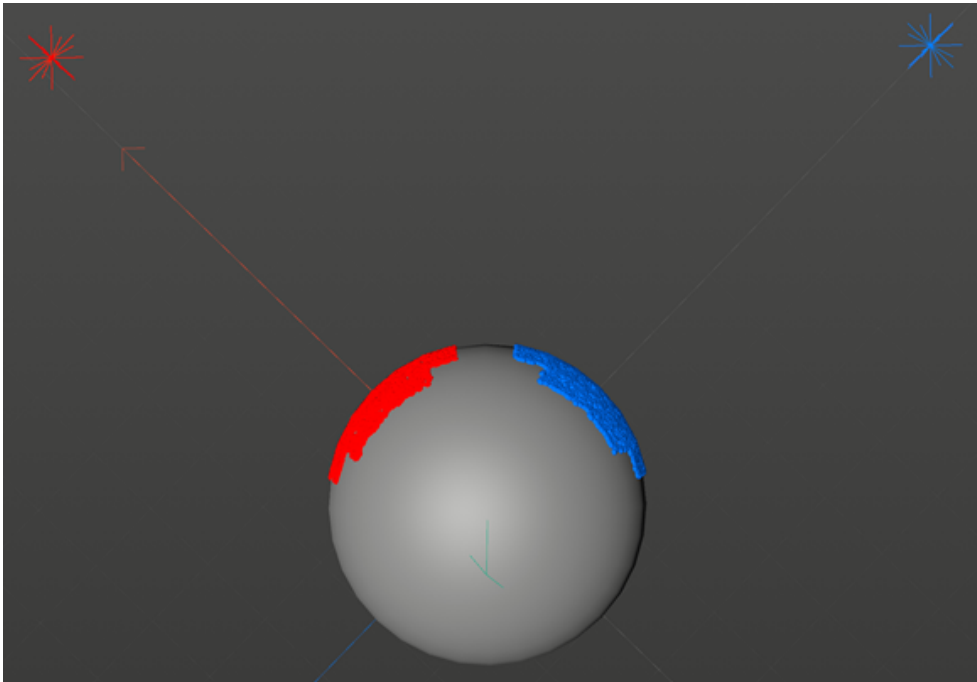



Figure 4.8. Illumination-controlled emission with higher 'Brightness Threshold'

4.2.7 Emitting from an object's position

 Rather than emitting from the polygons, points or edges of an object, you can emit simply from an object's position. You still need an object that the emitter can emit from (for example, a Null object won't work) but emission is only from the object's position in the 3D world. See [this animation](#) for an example. The scene file *cb4_emitter_objpos.c4d* for the animation is also in the chapter's download archive.

Objects which generate others, such as the Cloner, are special cases for this mode since they don't have a single position. How these are handled is explained in detail in the XP manual.

4.2.8 Emitting from an object's volume

This can be used to emit from the space enclosed by an object. The object has to be a polygon object and those without a volume, such as the Plane object, won't work (but you could give it some thickness with Cinema 4D's Cloth Surface object, and then it would work).

It can be quite useful to do this. It's particularly useful if you want to emit a mass of particles inside an object right at the start of the scene, and of course the particles will fill the shape of the object even if it isn't a regular object like a cube or sphere.

There's one additional parameter that is very important: **'Max. Tries'**. When the emitter wants to generate some particles, it has to find a location inside the object. For a cube or sphere, this is really easy but for a very irregular object, especially one with lots of little crevices, it can be very difficult to find a location. To prevent the emitter from trying forever, and locking up Cinema, this setting acts as a limit on the number of times it will try. If you find that an object isn't being filled as well as you would like, you can try increasing this value - but doing so will slow down the animation as the emitter will try for longer to find suitable locations.

4.2.9 Voxel grid emission

This is very similar to emitting from an object's volume but instead of just filling an object at any randomly-selected location inside the object, the emitter divides the object volume into a grid of small virtual cubes called voxels. Particles are then emitted at the centre of the voxel.

There are some options. You can choose to fill the grid randomly or do it in an ordered fashion, which will fill the grid from one side to another. The 'side' is one of the three axes, X, Y or Z and if **'Stop on Grid Full'** is checked, the emitter will stop emitting once each voxel has a particle. This means that in **'Ordered'** mode each voxel will receive one and only one particle, while in **'Random'** mode some voxels may get multiple particles and some may have none.

4.3 Other options

There are several additional options when emitting from an object. Not all of these apply to all objects, or to every source (polygons, points, etc.) of every object.

4.3.1 Particle direction

When a particle is emitted it must be given a direction. This is simple when a basic emitter such as a rectangle is concerned, but there are more options when emitting from an object.

As well as being able to emit along the orthogonal axes (X, Y and Z) of the object and in a completely random direction, you can choose to emit along the normal. What actually happens depends on the object and emission source. For polygon objects, either the polygon normal or the vertex normal is used, depending on whether you emit from polygons, points or edges (edges use the vertex normals). For splines, emission from points will use the vertex normals. Emission from a spline edge (which is really not an edge at all but the part of a spline between two adjacent vertices) is more complex. The direction is calculated so that, if the particle is emitted at the very start of the edge, the direction will point to the very end of the edge; if it is emitted from the very end, it will point towards the next vertex along the spline, if there is one. If there isn't the direction will be to the end of the spline for all particles emitted from that 'edge'.

An alternative to the normal is the phong normal. Phong normals can vary slightly from the face normal so you may find that with curved polygon objects the phong normal gives a better result. For emission from points, the phong normal is ignored and the vertex normal is used. Again, edges are slightly different. The phong normal is ignored for edge emission from polygon objects while for splines, the same method described above but with a different algorithm is used.

Figure 4.9 shows an example of the difference between normal and phong normal when emitting from a spline using edge emission. On the left (red) the direction is set to normal, on the right (blue) it is phong normal.

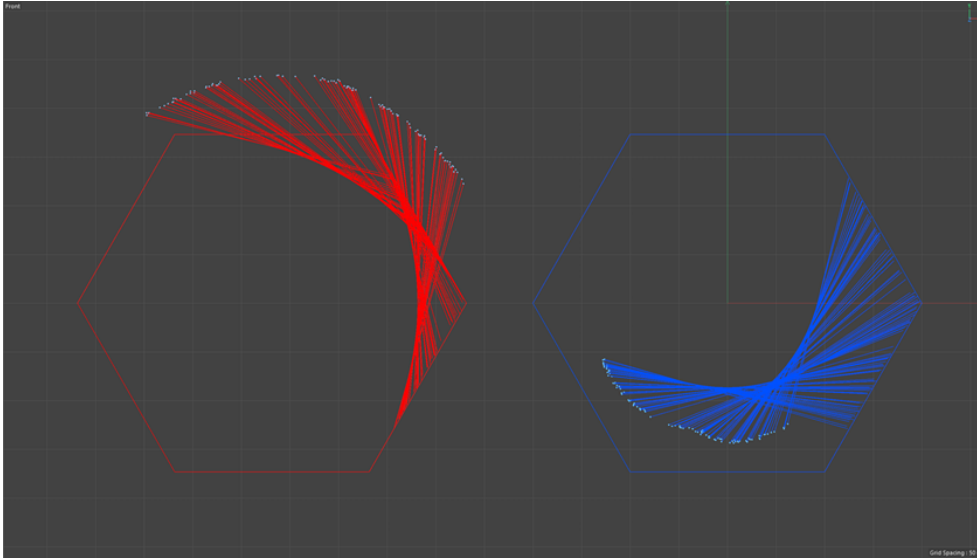


Figure 4.9. Particle direction derived from polygon normal (left) and phong normal right)

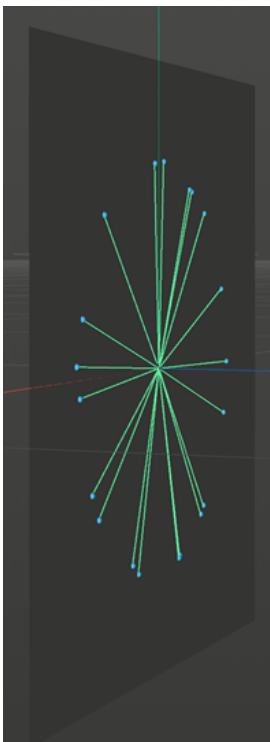
Regardless of the object and source, if you choose normal or phong normal you can invert the direction by turning on the **Invert** switch.

4.3.2 Additional controls for direction

You can see these by clicking the little arrow next to the **Particle Direction** menu.

Perpendicular to Normal

This will cause the particle direction to be perpendicular to the normal. Think about what this means. If a polygon normal is pointing along the +Z axis, what is perpendicular to that? It must be a direction for which the Z component is zero, but the X and Y components could be any value. The result is this, where the normal of this single polygon points along +Z. The direction gives no movement along Z, only X and Y:



This switch has no effect on point or edge emission or on splines at all. There is a another switch associated with this one - **Persist While Stuck** - which we'll discuss later on.

Figure 4.10. Particle direction perpendicular to the normal

Random Blend

This spline control is only available in certain modes (see the manual for details). What it does is blend the direction calculated from the surface normal, using the spline to determine the amount of blend. In this image you see the effect of blending with the spline set differently:

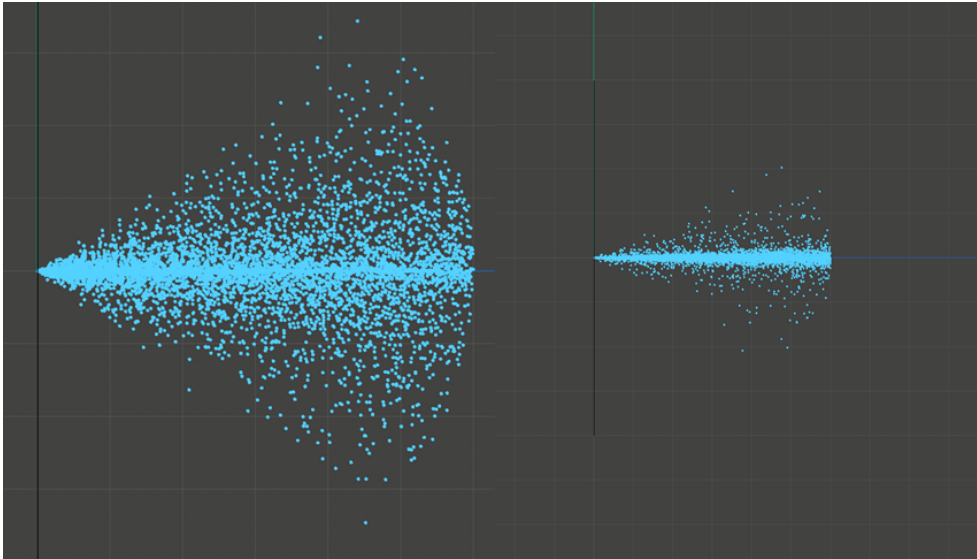


Figure 4.11. Random blend of particle direction with different spline control settings

These are the splines which produce these results:

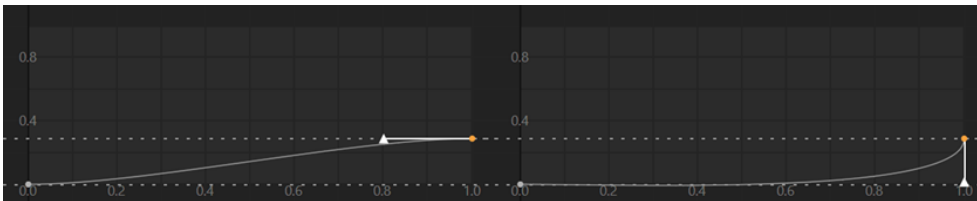


Figure 4.12. Splines producing the effects in Figure 4.11

4.3.3 Stick Particle to Source Object

This very important switch does exactly what it says. When turned on, any emitted particle will stick to the object at the location on the object it was emitted from. Don't make the mistake of thinking this is just a zero speed. If the object moves, the particle will move with it.


If a particle is emitted with the default speed of 150 scene units/second, but doesn't move because it is stuck, and the object to which it is stuck is moving with a speed of 30 scene units/second, what is the actual particle speed? It actually has two speeds! If you test the speed using a Question object or use the XP Console to see the speed - or use the Get Particle Data Xpresso node - all will show a speed of 150. And if you release the particle from its stuck position, it will move off with a speed of 150.

This is because what stops the particle moving is not a speed of zero but an internal flag in the particle which tells the emitter only to move the particle so it remains in the same position on the source object. But clearly, the particle is moving in the 3D world with a speed in this case of 30. How can you access that?

It's very simple: go to the **'Extended Data'** tab and turn on the **'World Speed'** switch. This causes the emitter to calculate the particle speed in the 3D world and stores the result in the particle's data set, updating it each frame. You can then test this with a Question object if you need to.

Let's go back to the direction controls and the **'Persist While Stuck'** switch. Suppose you have emitted particles which are stuck to an object and you've aligned the particle to point along the direction of travel, which is along the normal (we'll cover this in the chapter on extended data). Now you do something which is deforming the normals of the object - a Displacer deformer will do this. You'd like the particles to be re-aligned to the new normals when they change. The way to do this is to enable **'Persist While Stuck'**. All it does is force the emitter to update the particle rotation so that it remains aligned with the normal. This does mean that the

direction must be set to **'Normal'** - no other option will work - and you need to set up the rotations correctly, which are covered later in this chapter.

 The XP manual has a nice animation showing this working, which is so appropriate I've provided another version for this book, which is on the [book's website](#). On the left, the switch is off, on the right it is on. You can see that on the left the particle alignment doesn't change even when the normal does, but on the right, as the polygon normal is changed the particle is re-aligned to match it. The scene file [cb4_emitter_persistwhilestuck.c4d](#) is included in the download archive for this chapter.

4.3.4 Other controls

Set Radius by Polygon Size

This control probably isn't used much, but could be useful in some cases. If turned on, the particle radius varies according to the size of the emitting polygon. How does the emitter know what size to use? You have to provide a radius and, crucially, a variation value. If the radius is 3 and the variation 2, the smallest polygon will set a radius of 1 ($3 - 2$) and the the largest will set a radius of 5 ($3 + 2$).

Here's what you might see:

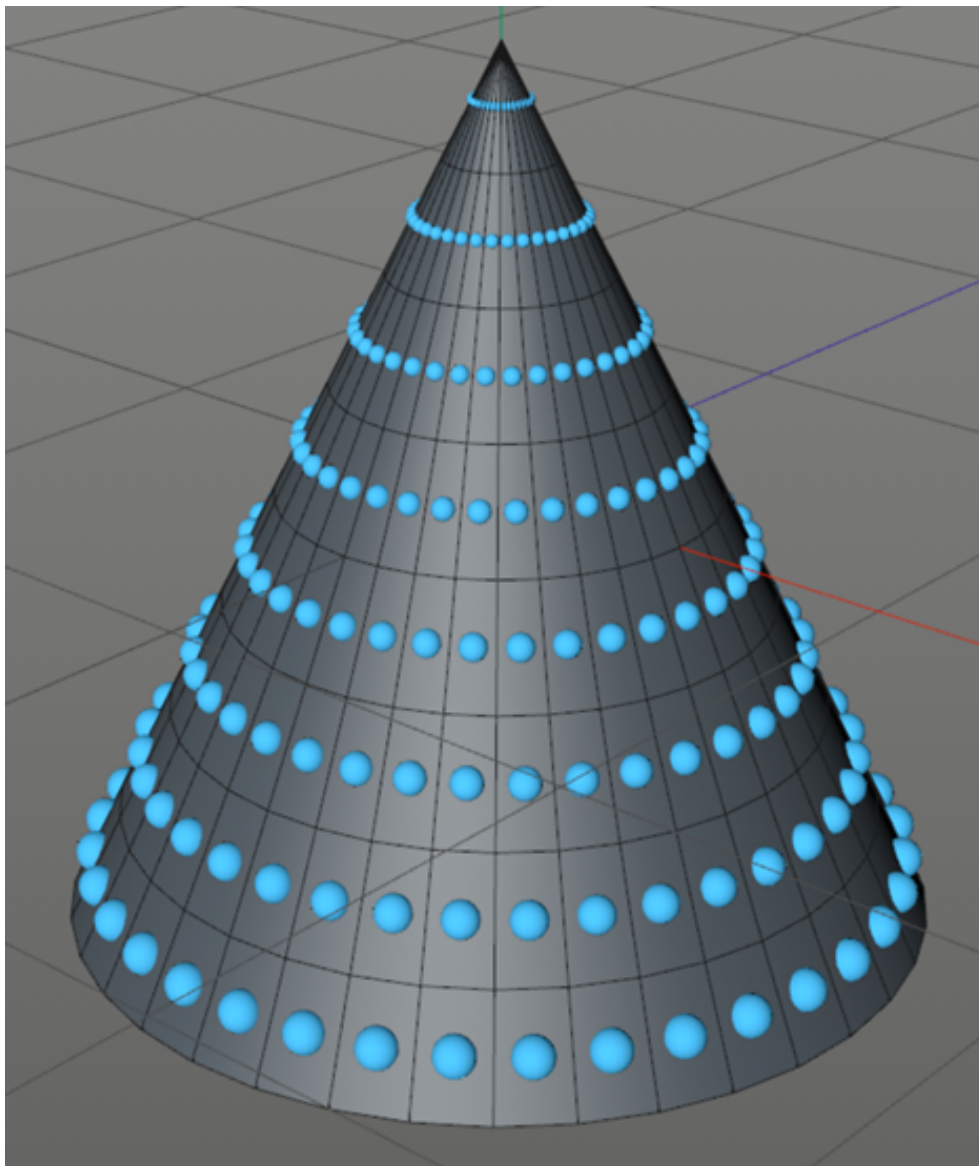


Figure 4.13. Setting particle radius from polygon size

The scene file [cb4_emitter_radiusbypolysize.c4d](#) which produced this is in the download archive for this chapter.

One Particle per Source Element

If this switch is turned on, the emitter will emit only one particle per source element. The 'source element' in this case means something which has a single, fixed location so that one particle can be emitted from that location. The only modes which fit this requirement are polygon centre, points, and the object's position. All other modes don't have a single, fixed position - for example, polygon area could be anywhere on the surface of a polygon.

No matter what the emission rate is, the emitter will emit one particle from each element every frame. Even if you set the birthrate to zero, particles are emitted and will continue to be emitted each frame (unless you are using shot mode, for example - then you will get one shot with one particle per element, even if you set the shot count to zero).

i There is a limitation here. Suppose you have a Cloner and want to emit one particle from each clone's position. This works fine, but what if you want to emit 10 particles per clone? If the emission mode is set to '**Object Position**' and you have enabled '**One Particle per Source Element**', a further control labelled '**Multiplier**' becomes available. If this is set to 10, it will cause 10 particles to be emitted per clone.

Object Chain

This is a really important control if you are emitting particles from a Cloner, or from a Null object which has a number of child objects to emit from. The default setting is '**Any Object**'. What this means is that the emitter will look at the list of objects and pick one at random. That object is used as the emitting object for that frame. Next frame, the process is repeated; the emitter may choose the same object or a different one. This may be fine and what you want to happen, but in some cases you might prefer to have the emission for each frame distributed across all the objects rather than using only one. You can do this by setting this menu to '**Connect Objects**' which does exactly that - it connects all the objects into one large object (internally of course - nothing changes in the actual scene) then emits from that single object.

You can also choose which of the objects to emit from. By choosing '**Use a Specific Object**' and then keyframing the '**Object Number**' control, you could have each object emit particles in turn. The download archive for this chapter contains a scene file [ch4_emitter_objchain.c4d](#) to show this.

4.4 Topology

The topology controls are available for the three polygon emission modes plus texture and illumination. They enable the control of emission depending on the height and/or slope of an emission location on the object.

Depending on the object and the settings used in this section, it could be impossible to find any suitable points and XP would appear to freeze. The '**Max Tries**' setting is there to prevent this. You can increase that value if insufficient particles are being emitted, but it will inevitably take more time and slow the playback.

4.4.1 Height

In this context, height really means the distance along an axis. If this is the Y axis (the default) then it really does look like actual height, but you can use the X or Z axis instead.

But what distance is being measured? By default, this works in local space - that is, local to the object. So the distance is the distance away from the object axis. This isn't always convenient. For example, if you are emitting from a Landscape object, the axis is in the centre of the object, not at the base as you might expect - and you can't alter it. Instead, you can change to world space using the '**Space**' menu. The distance is then from the 3D world centre, so you could move the Landscape so its base was at zero on the Y axis and now the distance is the actual height above the world centre.

Having chosen the space and the axis, to use height just enable the '**Use Height**' switch. In Figure 4.14, the Landscape object is 125 units along the Y axis, world space is used and the base of the object is set at zero on the Y axis. The minimum and maximum height values are set at zero and 100 units respectively.

You can see that no particles are emitted over the 100 units height value. This works fine but looks unnatural - it certainly wouldn't look realistic if these were trees growing on a hillside, for example. To fix, this, you can use the '**Spline**' control, which alters the distribution of particles within the height range. See Figure 4.15 for the altered spline and Figure 4.16 for the result.

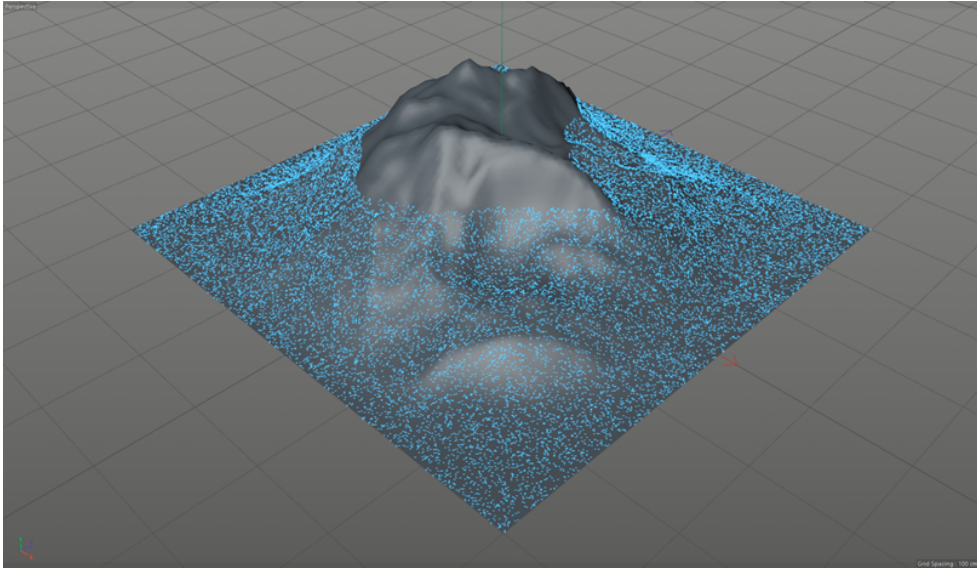


Figure 4.14. Height-based emission (note the sharp cut-off)

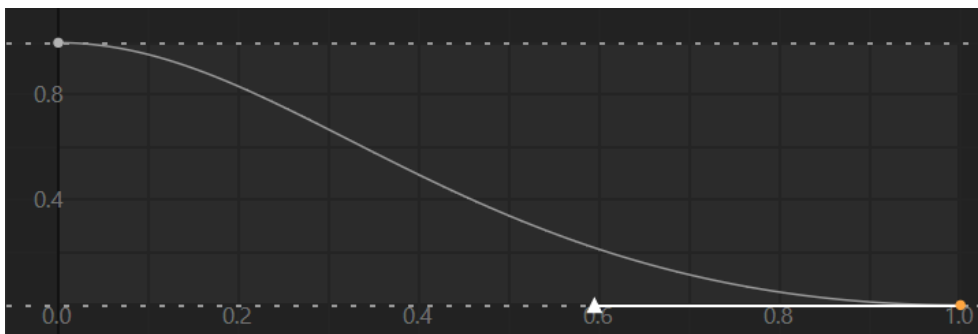


Figure 4.15. Spline to produce a feathered cut-off for emission

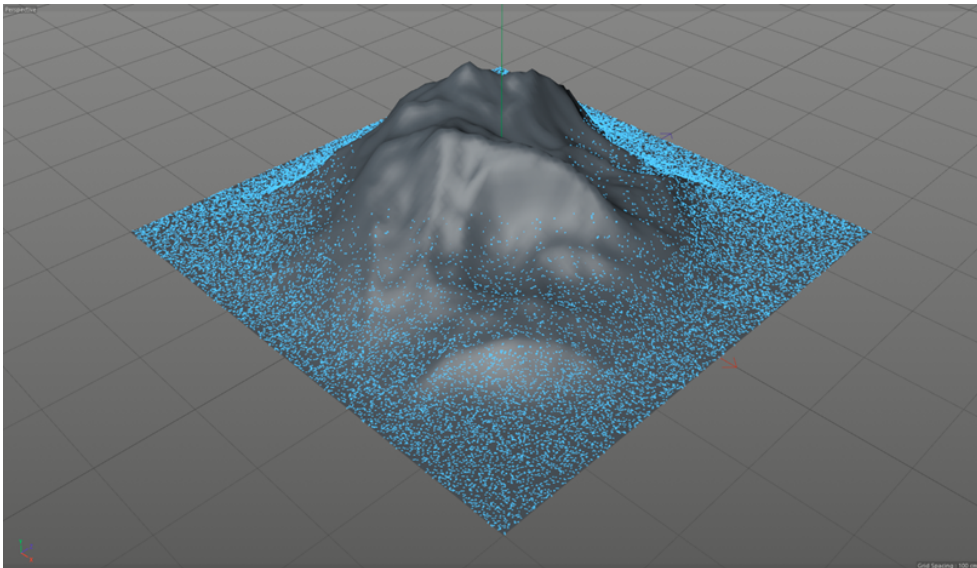


Figure 4.16. Same scene as in Figure 4.14, but using the spline control from Figure 4.15

Figure 34 shows that particles are emitted less densely as the height increases, which looks much more natural. This scene file is in the download archive as [ch4_emitter_topology_height.c4d](#).

4.4.2 Slope

Slope works in exactly the same way as height, except that the range is of angles, not distances. Ninety degrees on the Y axis would be vertically upwards. With the **'Max Slope'** set to 30 degrees, with the same landscape object we would see this:

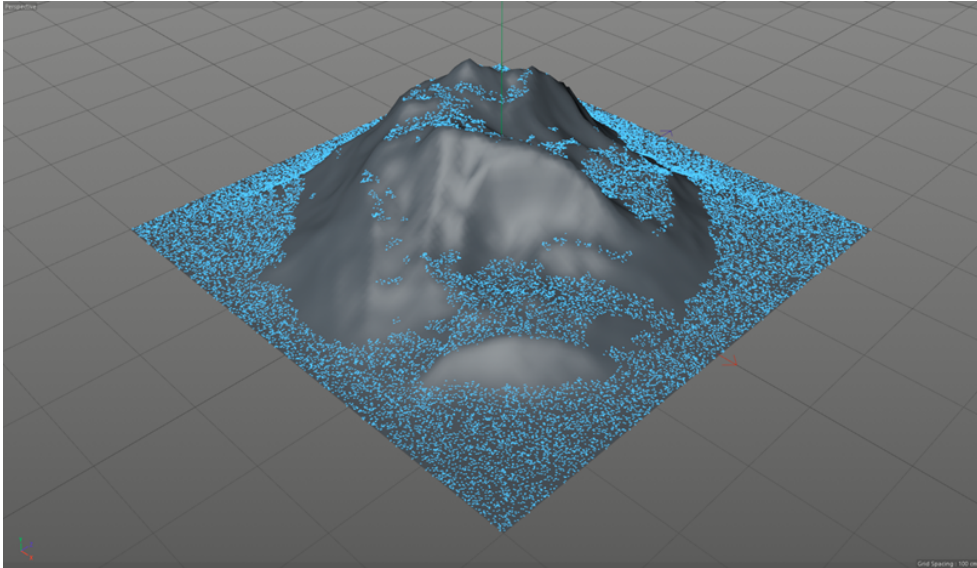


Figure 4.17. Slope-based emission

You can see that particles are not emitted on the steeper parts of the object. Slope has its own spline control, which can be used in the same way as height to feather the distribution across the minimum/maximum slope range.

4.4.3 Curvature

The curvature of any point on the surface of an object can be calculated from the angle of an invisible line from that point to any nearby polygons. Consider a flat surface such as a Plane object. Because it is completely flat, the angle from any point on the surface to any nearby point will never be less than 90 degrees. The same would be true of a sphere or a cube or in fact any object that doesn't have any crevices or ridges in the surface. Using curvature to control emission on such an object will cause no particles to be emitted.

Objects such as the Landscape object do have such folds in the surface topology and are very suitable for this technique. Using the same object as in the previous examples and the default curvature settings we get this result:

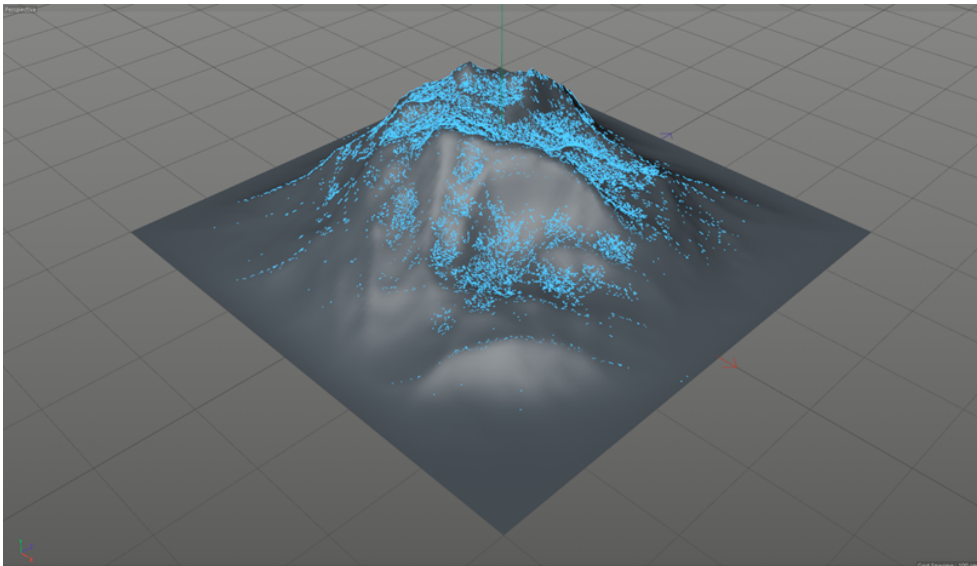


Figure 4.18. Curvature-based emission

You can see that the particles are emitted from areas where there is a degree of folding to produce crevices and ridges. The '**Ray Angle**' setting is the maximum angle between the emission point and any neighbouring polygon. Particles will only be emitted if the angle is less than this value. Lowering the value will restrict emission to narrower crevices and sharper ridges. With a value of 40 degrees, we see the effect in Figure 4.19.

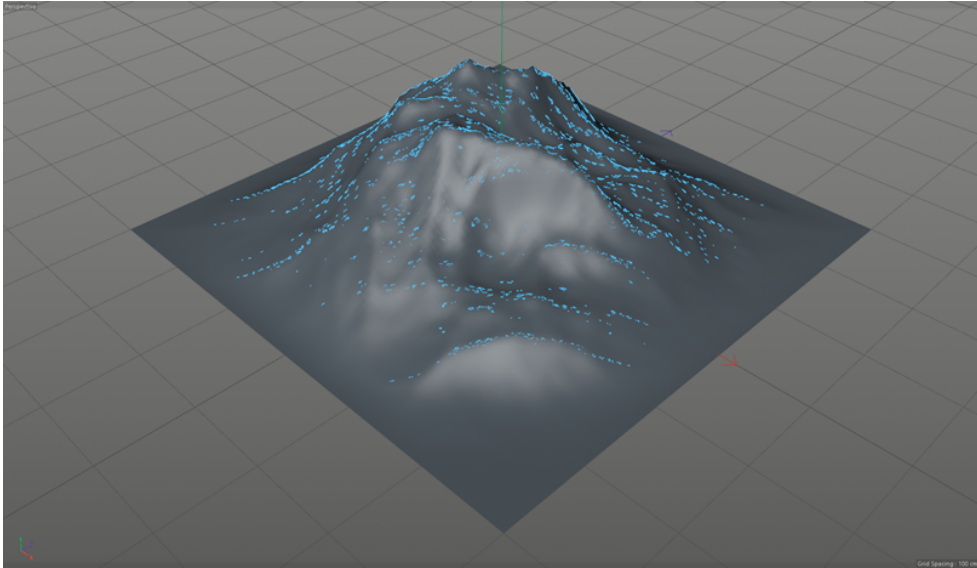


Figure 4.19. Same scene as Figure 36 but with a lower 'Ray Angle'

This can be very effective but the smaller the angle, the harder it is to find a suitable emission point, so this will take longer to perform.

The '**Ray Length**' setting is the length of that 'invisible line' referred to above; that is, a measure of the distance between a surface point and any neighbouring polygons with a suitable angle to that point. With a ray angle of 80 degrees and a ray length of 5, we would see this:

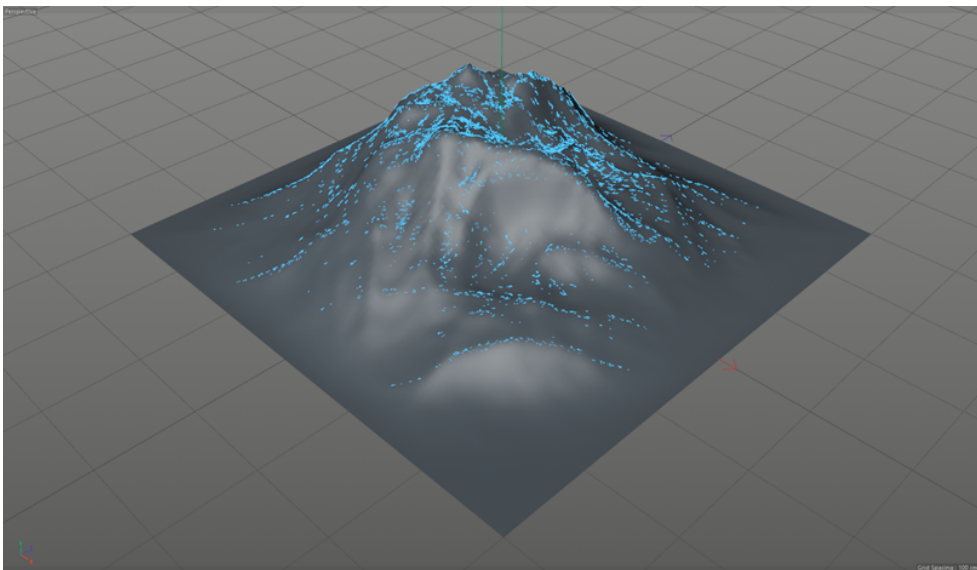


Figure 4.20. Same scene as Figure 36 but with a lower 'Ray Length'

This has the effect of restricting the emission to narrower folds in the object.

The example file [cb4_emitter_topology_curvature.c4d](#) is in the download archive for this chapter.

With small angle and length settings, it may be very difficult to find a suitable emission point. As mentioned above, you could increase the 'Max. Tries' value to increase the number of attempts, but when using curvature there's a better way.

You could increase the chance of finding a suitable emission point by increasing the number of those invisible lines used to detect neighbouring polygons. These are termed 'rays' and are simply lines sent out at random directions from a surface point which may then intersect with nearby polygons. Clearly, the more rays you have the greater the chance of finding a polygon within the ray angle and ray length limits. However, this will certainly take longer to carry out, giving slower playback.

4.5 Emitter falloff

The emitter has its own inbuilt falloff settings. These are not the same (they're much simpler, for one thing) than falloffs in pre-R20 versions of Cinema 4D or fields in later versions.

Falloff can only be used when emitting from an object and then only in some modes.

4.5.1 Using falloff

By default the emitter falloff is set to **'Infinite'** which has no effect. To see any change you must one of the other modes. The shape can be either a sphere or a box. You can resize it but you can't move it independently of the emitter. That doesn't matter though; since particles are being emitted from an object, the position of the emitter itself is irrelevant. So to move the falloff, simply move the emitter.

4.5.2 Falloff modes

Next, select the mode. There are three options:

Boundary

If you select this, particles are distributed evenly within the bounds of the falloff shape. For example, a Sphere falloff set to **'Boundary'** would give this result:

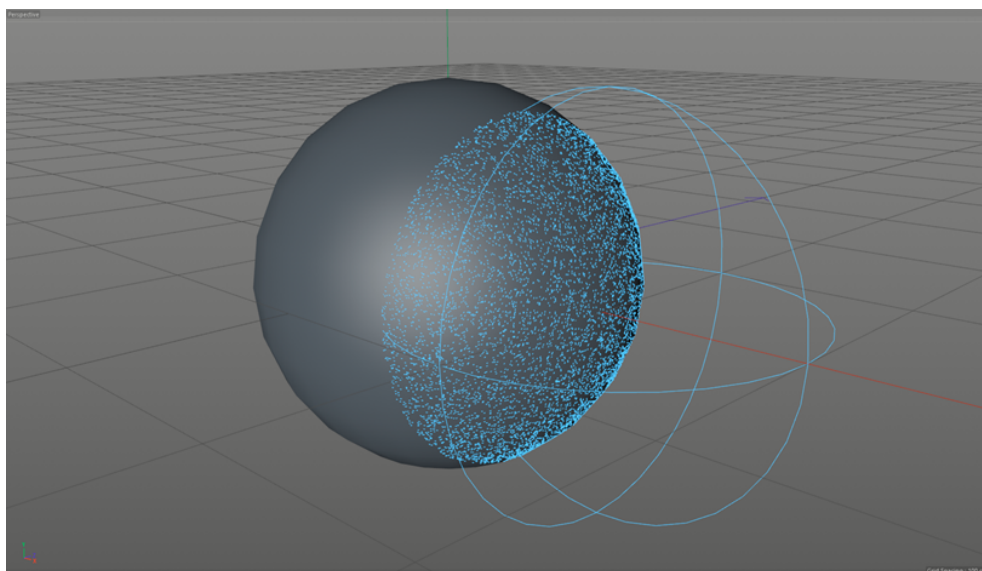


Figure 4.21. Emitter falloff 'Boundary'

Linear

With this mode, the chances of a particle being emitted decrease the farther the emission point is from the centre of the falloff. This gives a feathered edge to the area of emission as shown in Figure 4.22.

Spline

This mode uses a spline control to distribute the particles. By default it gives results which are very similar to linear mode, but adjusting the spline can give some interesting effects, demonstrated in Figure 4.23.

The spline which produces this effect is displayed in Figure 4.24.

A file to demonstrate these effects is in the download archive as [cb4_emitter_falloff.c4d](#).

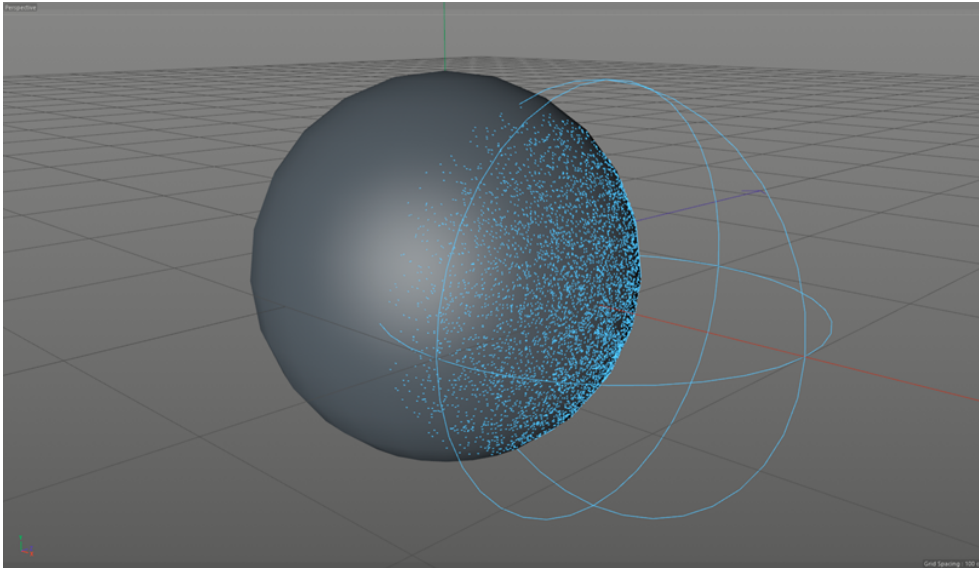


Figure 4.22. Emitter falloff 'Linear'

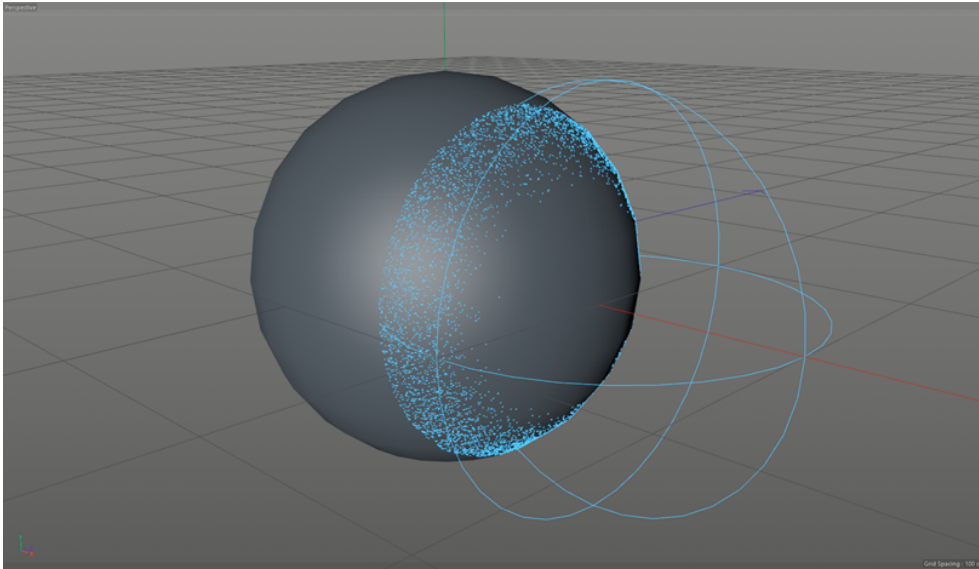


Figure 4.23. Emitter falloff 'Spline'

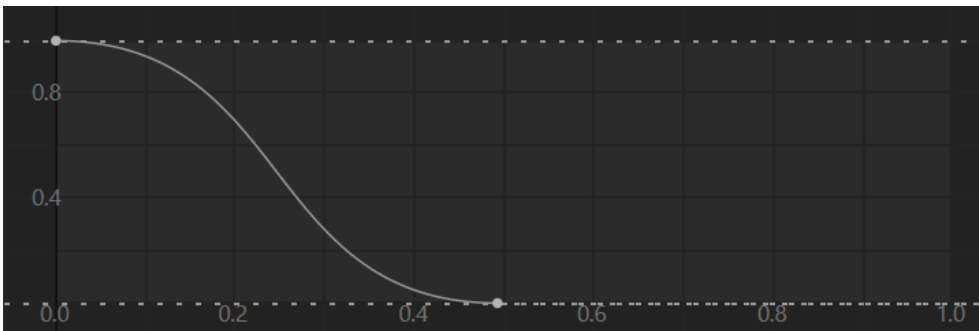


Figure 4.24. The spline giving the effect in Figure 4.2.3

Chapter 5: The Emitter: Extended Particle Data

As mentioned in Chapter 1, the particle cannot carry every possible item of data with it just in case it might be needed. This would add enormously to the memory used for each particle, so some data is only added when actually needed, and then memory is allocated specifically for it. In the emitter this is called extended particle data, but in fact many other objects, especially modifiers, also allocate extended data for their own use. In most cases, however, users never see this, although it is sometime exposed for use in the Question object. In the emitter, you can see and add this data when required.

↓ The scene file(s) for this chapter can be found in the archive [examples_ch5.zip](#) on the book's website. Click here to [download the archive](#).

5.1 Particle rotation

5.1.1 Enabling rotation

Rotation data is needed less often than might be thought, so it is a good candidate for extended data to be added when needed. If you want to rotate particles then you must add this data by enabling the **'Use Rotation'** switch in the emitter. However, some other X-Particles objects will add the rotation data themselves for convenience. For example, if you add a Spin modifier to the scene, it will work without your having to enable rotations in the emitter. It doesn't matter if you do enable them, it has no additional effect, but it isn't necessary.

Once you enable rotations, there are a number of options available. The most important is the **'Rotation'** menu, which controls what the emitter will do to rotate particles.

⚠ This is an important point: the menu only governs what the emitter will do. It doesn't affect what other objects may do. For example, setting it to **'None'** does not mean that particles cannot rotate. It simply means that the emitter will not rotate them, but other objects may do so.

As already mentioned, some objects will add the necessary rotation data themselves so they can rotate particles, but not all do so and in those cases you must still enable rotations so that the data is added - even if the emitter does not do anything with it.

'Random' means that the emitter will give the particles a random rotation when they are created. After that, it won't affect the rotation at all. You can choose the axis or axes to rotate around in the **'Random Rotation Axis'** control. **'Set'** is equally simple: you specify a value around one or more axes in the **'Rotation'** setting and the particles are rotated accordingly - but again, only on creation and not thereafter.

5.1.2 'Facing' something

There are three modes which make the particle face towards something. **'Face Camera'** points the particle towards the active camera. It does this by calculating a vector from the camera to the particle, then changing the particle's Z-axis to point along the reverse of that vector. This is important, because doing that potentially changes the rotation on all three axes, and the particles will all have slightly different rotations depending on their position in the 3D world. The alternative is **'Face Screen'** which makes the particle face the viewport but always with rotation on the bank (B) axis set to zero; all particles will have exactly the same rotation regardless of position.

Here are examples showing the difference. First, **'Face Camera'** is shown in Figure 5.1.

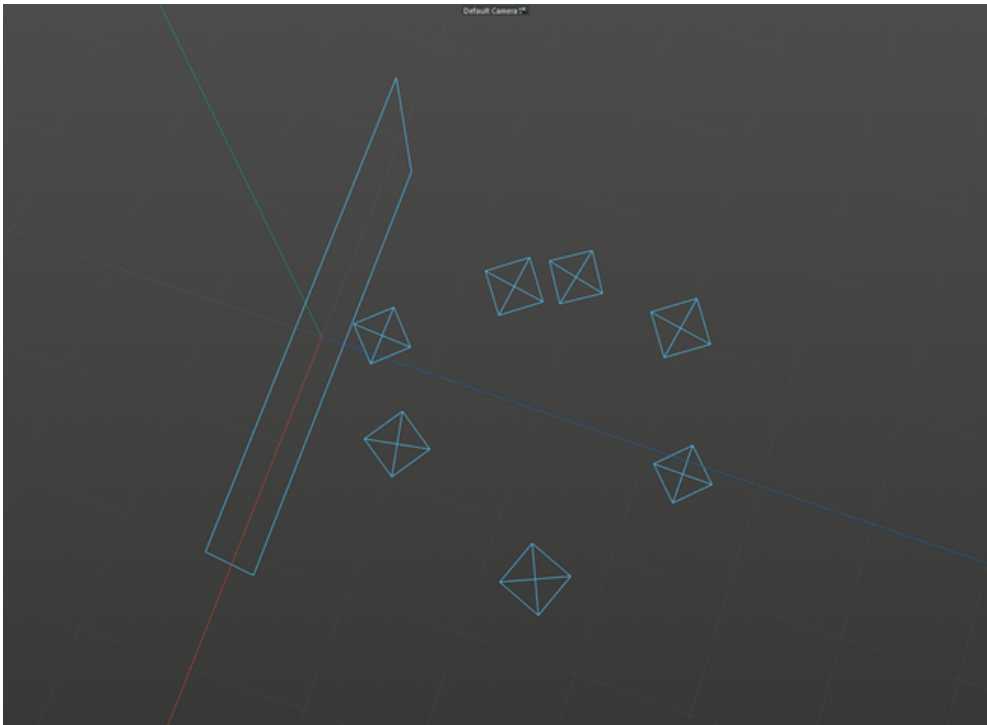


Figure 5.1. Rotation set to 'Face Camera'

Note that the particles all look at the camera but appear to be rotated differently; this is because the rotation needed to point at the camera will be different if the 3D world position of the particle is different. We can see this by looking at the rotation data in the XP Console (Figure 5.2), where the rotation angles for each particle are different:

X-Particles Console

Data to Display

Particle Data

Options

Emitter for Output

Emitter

xpEmitter

Get Active Emitter

☒ Lock Emitter

PID	Age	Rot (H)	Rot (P)	Rot (B)
1	35	248.05	77.39	65.83
2	30	261.74	72.01	42.68
3	25	269.94	84.77	68.78
4	20	278.03	71.32	24.26
5	15	286.59	72.05	19.47
6	10	311.42	78.25	14.25
7	5	307.36	73.48	5.17

Figure 5.2. Rotation data for 'Face Camera' mode showing each particle has a different rotation

For **'Face Screen'** see Figure 5.3. The particles look as if they all have the same rotation, which we can show in the console is the case (Figure 5.4).

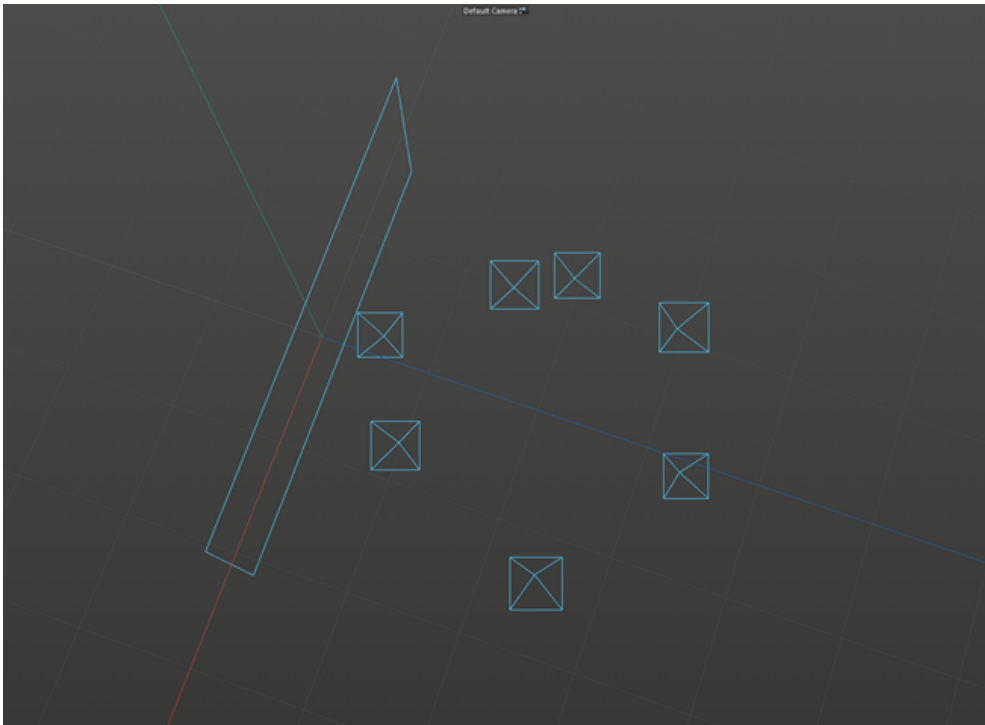


Figure 5.3. Rotation set to 'Face Screen'

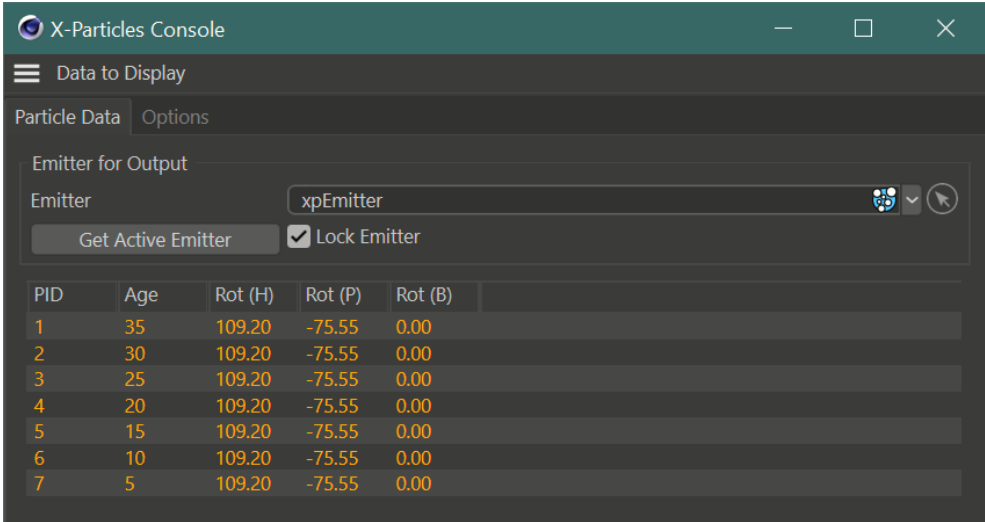


Figure 5.4. Rotation data for 'Face Screen' mode showing identical rotation for each particle

The third facing option is **'Face Object'**. This is the same as **'Face Camera'** except the particles will be rotated to align their Z-axis with the specified object (which can be any object in the scene).

5.1.3 Tangential rotation

This is an important mode which rotates the particle to align it with its direction of travel. It's used when you have a moving particle that you would like it always to point towards the way it is going. For example, an arrow which always points along the Z axis while the particle itself is moving along the X axis is not going to look very good. This is where tangential rotation comes in.

This is the effect you get in this mode (Figure 5.5). A Trail object has been added to show how the particles are moving. Note how the arrows are all rotated to follow their direction of travel:

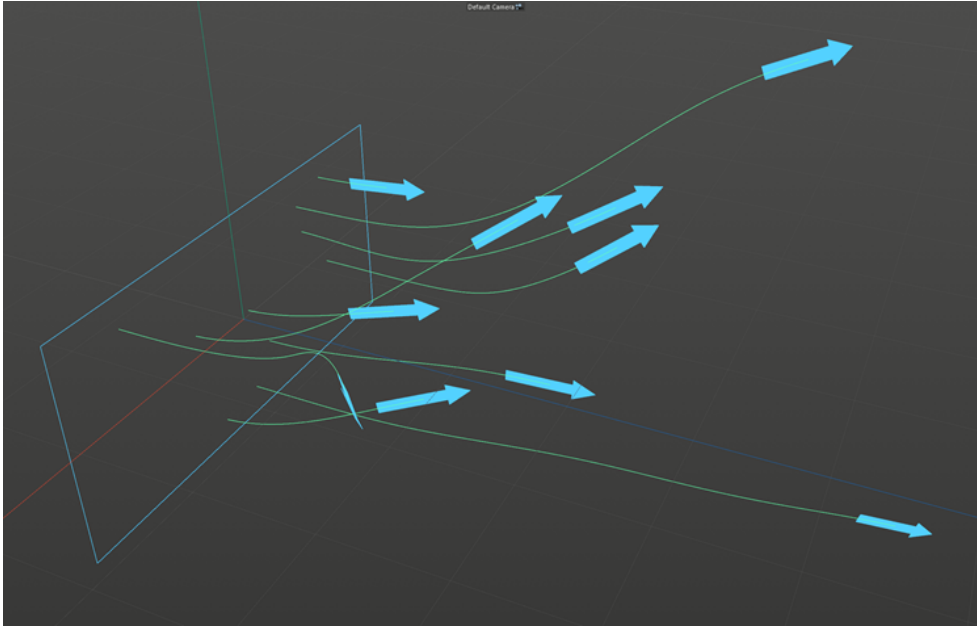


Figure 5.5. Tangential rotation

There are a number of options. Most of the time you will want the particle Z axis to point along the direction, but you don't have to. It can be the particle X or Y axis instead. For example, choosing the Y axis will show this:

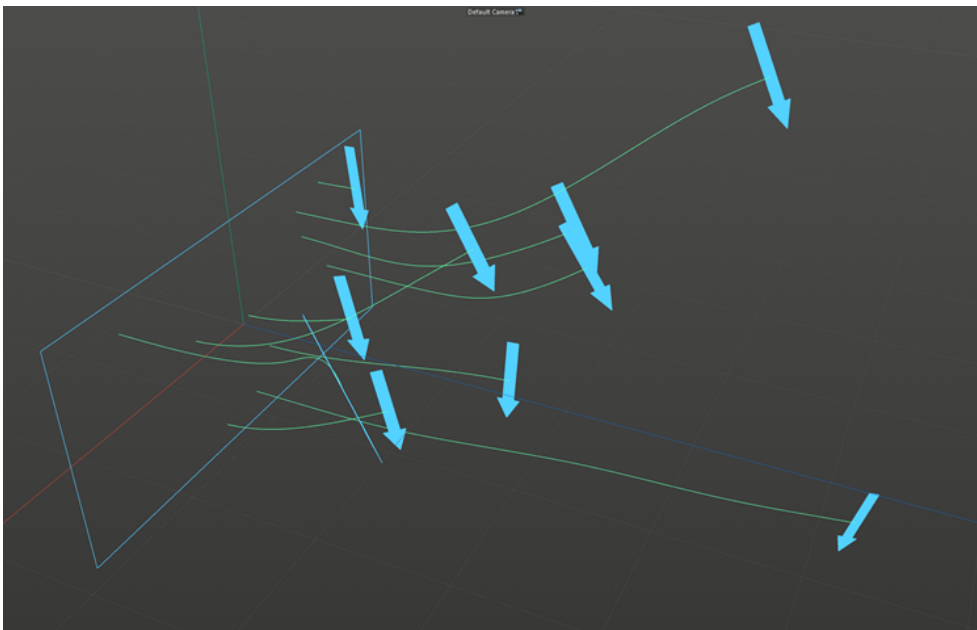


Figure 5.6. Tangential rotation but with particle Y axis pointing along the direction of travel

You can see that the Y axis of the particle now points along the particle direction. Most of the time you won't need this, but think what happens if you use an X-Particles Generator object to link an object to each particle. If the tangential axis is the particle Z axis, the Z axis of the particle points along the direction. If the object being generated has a definite 'point' (e.g. the nose of an aircraft) you would probably want that to point along the direction. Figure 5.7 shows what can happen if the 'nose' isn't aligned along the object's Z axis.

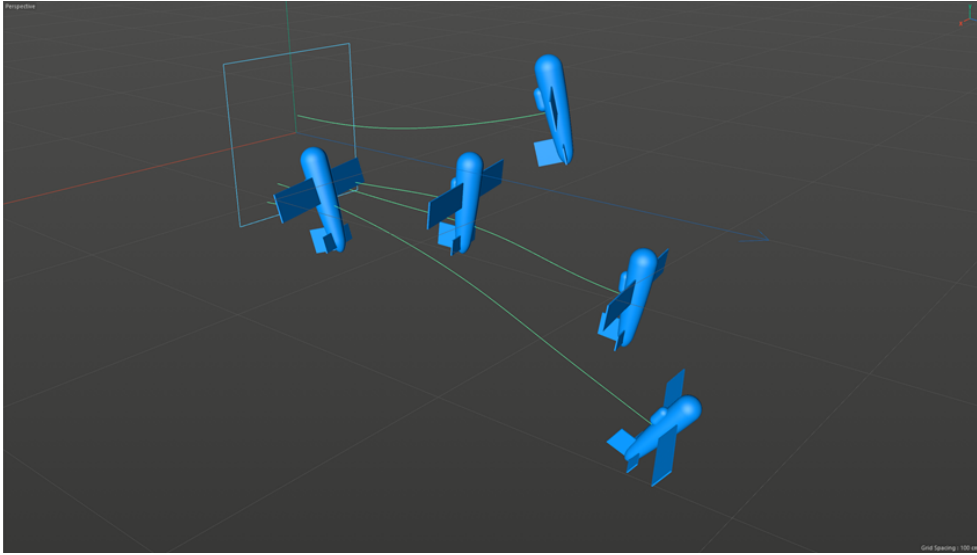


Figure 5.7. Incorrect alignment of particle along Z axis for tangential rotation

That's clearly not right. It occurs because the aircraft's nose points along the object's Y axis. Sometimes you can solve this simply by changing the object's orientation, as with primitives such as Cone, Pyramid, etc. They have a control where you can set the orientation to the positive Z axis, but most objects won't have such a control. You can try rotating the object's axis, but a simpler way in this case is to change **'Tangential Axis'** to match the object - that is, change it to the Y axis. Doing that corrects the problem:

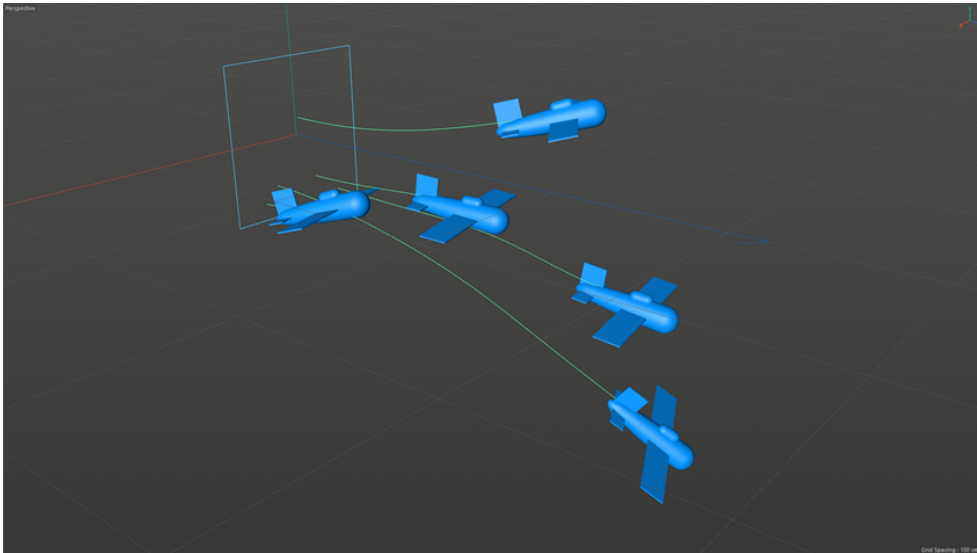


Figure 5.8. Corrected alignment for tangential rotation (this time along the Y axis)

This file is included in the download archive for this chapter as [*ch5_emitter_rotation_tangential_axis.c4d*](#).

You can opt to rotate the particles tangentially only on emission and not maintain that rotation afterwards by disabling the **'Persist Orientation'** switch. The **'Rate of Change'** setting controls how quickly the particle will be rotated to meet any change in direction. At 100%, the rotation will be fully updated each frame. Anything less will cause the particle rotation to be updated more slowly, so if you need the rotation changes to lag behind directional change, you can reduce this value. Finally, you can set an up-vector. Sometimes, when the particle rotates to align with a new direction, it will bank as well as changing the heading and/or pitch. This is natural behaviour (think of an aircraft turning - it will bank as well as changing its heading) and is what you want in most cases, but if you don't want it, set the 'Up-Vector' to an option other than **'None'**.

5.1.4 Up Vector

The final rotation mode is **'Up Vector'** (don't confuse this with the 'Up-Vector' setting in tangential rotation). This is a special case which is only used when you are using the Follow Surface modifier. This modifier will move particles over the surface of a polygon object and will change the particle direction in doing so. The problem is that if rotations are enabled the particle may suddenly 'flip' its

rotation, especially on the B (bank) axis, which looks poor. If this happens, use this rotation mode which will force the particle to align its Y axis to be perpendicular to the surface.

5.1.5 Particle spin

So far, we've just rotated particles - the rotation may or may not change but there is no spin, that is, no regular rotation each frame around one or more axes. You can do this by enabling the **'Simple Spin'** switch. This is an older feature of X-Particles which existed before the dedicated Spin modifier was added.

In all cases I would recommend that you use the Spin modifier. **'Simple Spin'** is there for compatibility with older scene files, but is deprecated and future versions of XP may lose it. If you do use it, simply enable the switch and set the amount of spin you'd like on one or more of the axes. You can add variation to this so different particles spin at different rates.

5.2 Other data


This section contains an variety of miscellaneous particle data. They can all be added by enabling the respective switch.

5.2.1 Rewind

This is data required by the Rewind modifier, which is not present in later versions of XP. This is no longer available but the switch is present for compatibility with older scenes and can otherwise be ignored.

5.2.2 UV Emission Data


If a particle has been emitted from an object which has UV data, there will be UV coordinates for the location of the particle emission. These can be stored with the particle as extended data if required. Note that for this to work properly the object must have correct UV coordinate data

 What can this be used for? One thing you can do is sample a texture on an object at that UV location whenever required. First, take a look at [the animation](#) on the book's website. This uses a more advanced technique - data mapping - which we'll cover in detail in another chapter. In the scene, a shot of particles is emitted from a Plane object using **'Polygon Area'** mode. The texture on the Plane, an animated noise, isn't used to control emission of the particles. What we want is to have the particle radius change so that it is zero on black areas and maximum size (set at 3 units here) in white areas. Grey areas will give the particles a size between these values. Since the noise is animated, we want the particle radius to change as the animation is played.

In the chapter on data mapping we'll dissect this scene but for now, what we are doing is mapping the particle radius to the brightness of the texture. Each frame, for each particle the texture is sampled at the UV coordinates stored in the particle. These coordinates won't change but the colour at those coordinates might do so. Internally the brightness is calculated as a value from 0 to 1, and then this is multiplied with the maximum radius value, which is 3. This gives us a radius from 0 to 3 which will vary as the texture varies on the object. The file is in the download archive as [ch5_emitter_uv_emitdata.c4d](#).

5.2.3 Nearest Particle Data

For each particle this data item stores the distance to the nearest and furthest other particles. You can test this data in a Question object; for example, you could use this to trigger an Action if the nearest particle is closer than a value you specify. In the emitter display modes, it can also be used to control the particle colour based on how far away other particles are.

 We'll look at this later on but for now, see [the animation](#) here. This shows that particles are initially dark blue when emitted because they are close to other particles, but as they become separated the colour changes to light blue and then white. The scene file [ch5_emitter_nearest_ppdata.c4d](#) is in the download archive for this chapter.

5.2.4 World Speed

This was discussed in section 4.3.3 **'Stick Particle to Source Object'**. Please refer to that section for details of the difference between particle speed and world speed. Note that to use world speed, for example in a Question object, you must first enable this switch.

5.2.5 Emission Vertex

This switch is only available when emitting from an object's vertices. It stores the number of the vertex the particle was emitted from, and can be used in data mapping when mapping to vertex weight or a vertex map tag.

5.3 Physical Data

These extended data items are used for scenes with fire and smoke, and in the case of particle mass, also with dynamic engines such as xpBullet. We will look at those when we come to look at Explosia FX.

5.4 Fluid Data

As with physical data, the fluid data are used in the Fluid FX object and will be dealt with we look at X-Particles fluids.

5.5 Custom Data

As well as all the other data a particle can store, it's also possible to add your own custom data to all the rest. Why would you want to do that? The easy answer is that it's limited by your own imagination, but to give a taste of what can be done that you couldn't otherwise do, let's look at a (fairly simple) example. This does use some objects and concepts not yet covered, but they are essential to make this work.

In this example, what we want to do is change the colour and shape of the particle at some point in its lifetime, and this should be a randomly selected time for every particle. There is in fact no way to do this in XP unless custom data is used.

5.5.1 Setting up the data

The first thing is to set up the custom data. Let's say we want the change to be made at a point when the particle is somewhere between 20 and 60 frames old. What we will do is give each particle a random integer value which must fall between those limits. To do this, in the **'Custom Data'** quicktab, click **'Add'** to create a new custom data item. This will appear with various defaults; it's already an integer so we don't need to change that.

Since we are going to change and test the value of this data item, we have to be able to identify it. We can do this by using the ID value and/or giving it a name. The ID value is zero by default, and if we had more than one data item and were relying on the ID value alone, we would need to make sure that all the ID values were unique. For this reason I much prefer to name each item, especially since if you have multiple items it's so much easier to know which one you are dealing with. So we'll give it a name in the **'Name'** field, which in this case could be 'Countdown'.

Now we need to set the initial value - the one the particle is given when it is created. This should be set to 40. Now each particle will receive this data item with the value 40 on creation. However, we want a range of values. The way to add this is to change the **'Variation'** value. This is a percentage. If we make this 50%, the value on emission will be 40 +/- 50%, that is 40 +/- 20, which gives us the range we want of 20 to 60. Each particle will get a random value somewhere in this range. The interface should now look like this:

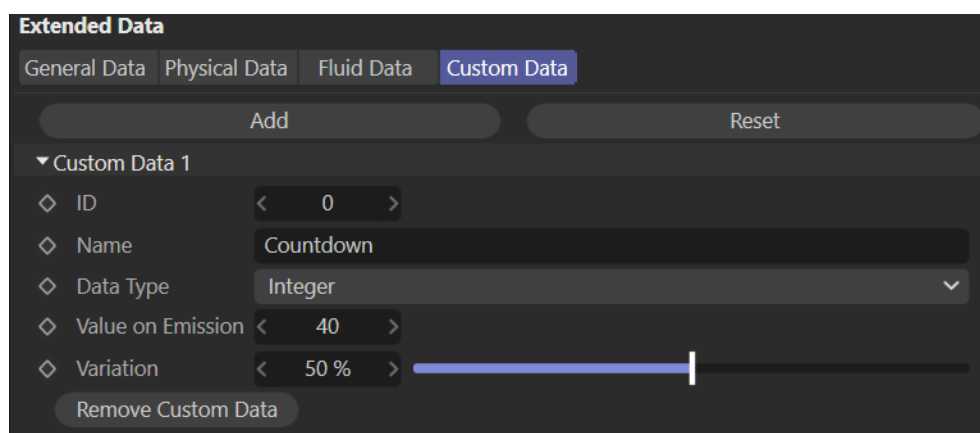


Figure 5.9. Adding custom data to particles

5.5.2 Changing the data

So far so good, but this doesn't do much. The value will never change so how can we test if the point at which the colour is to change has been reached? To do this we will need to decrement that value each frame so that eventually it reaches zero. (You could instead increment the value and test when it reached a specific value, such as 100. This works because the initial value will vary between particles so the lower the initial value, the longer it would take to reach 100.)

Decrementing (or incrementing) the data is an ideal task for a modifier, which will be called every frame. There is a dedicated modifier for this - the Custom Data Modifier. We'll look at this in more detail in a later chapter, but for now, we can add a modifier to the scene, specify which data item we want to work with and what we want to do with that data. The modifier should look like this after we set it up:

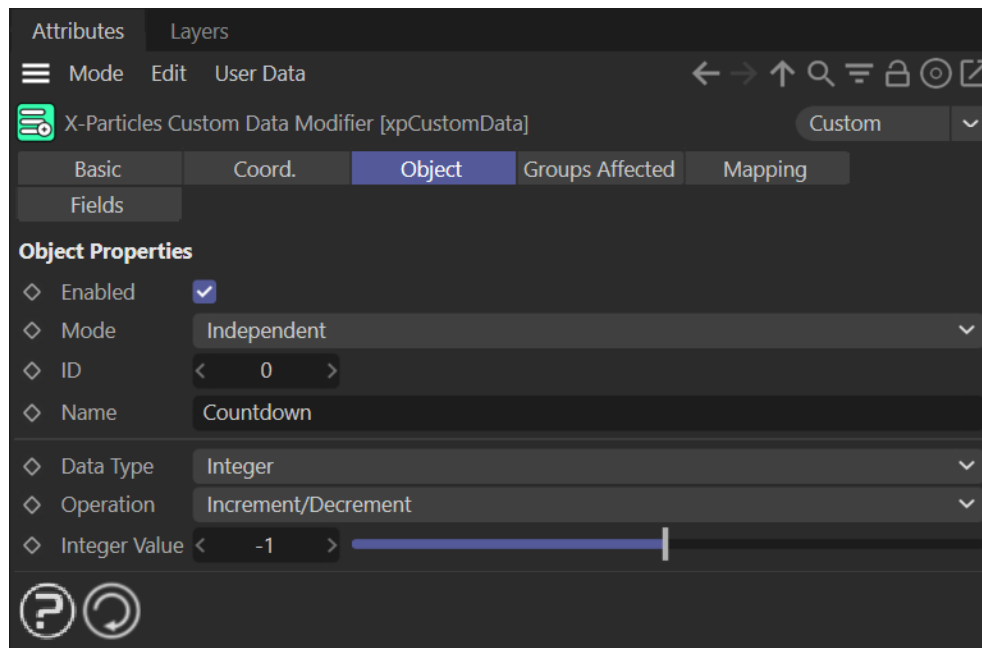



Figure 5.10. Adding a Custom Data Modifier

5.5.3 Testing the data

All we need to do now is check the data item value each frame and when it hits zero, change the colour and particle shape. You can do this with the Get Custom Data Xpresso node, but then you have to change the particle colour and shape also using Xpresso. It's much easier to use a Question object.

We'll look at Questions in a later chapter, but for now, all we need to do is add a Question object to the emitter and change it to test custom data. As with the modifier, we need to specify which custom data item we are working with - now you should see the reason that naming these items is so useful. Then we set up the test to check if the data item is zero, and if it is, call an Action which changes the particle shape and colour. Again, we'll look at Actions later on. The Question object would appear as shown in Figure 5.11.

Here we are testing the 'Countdown' data and we want to know if it equals zero. If it is, the 'Editor Display' action will be triggered.

 You can see the [resulting animation](#) on the book's website and the scene file [ch5_emitter_customdata.c4d](#) is in the download archive for this section.

Custom data is very powerful and should be considered whenever you need to test or change some kind of data which doesn't exist in XP.

Attributes

Layers

Mode

Edit

User Data

X-Particles Question Object [Custom Data]

Custom

Basic

Coord.

Object

Object Properties

Choose Question

Question Type

Particle Data

Particle Data

Custom Data

Random Seed

< 0 >

Question: Particle Custom Data

ID

< 0 >

Name

Countdown

Mode

Equals

Data Type

Integer

Value

< 0 >

Range

0

Sub-Questions

Test Sub-Questions Mode

AND

Create Sub-Question

Actions

Actions >

Editor Display

✓

Add Action

Activate/Deactivate Modifiers

Modifiers to Activate >

Figure 5.11. Testing custom data with a Question object

Chapter 6: The Emitter: Particle Display

Being able to visualise the particles is essential and X-Particles has many ways in which you can do this. In this chapter we'll look at some of these and touch on how you can render what you see in the viewport.

↓ The scene file(s) for this chapter can be found in the archive [examples_ch6.zip](#) on the book's website. Click here to [download the archive](#).

6.1 Showing the particles

6.1.1 Emitter colour vs. particle colour

In the emitter's **'Display'** tab you can see that there are two colour fields. One is labelled **'Icon Color'** and the other **'Particle Color'**. The icon colour is simply the colour of the emitter when drawn in the viewport (it won't be used if the emitter is not drawn, such as when emitting from an object).

The icon colour is really there just so you can distinguish between multiple emitters in your scene. It doesn't normally have any effect on the particle colour, unless you change the **'Color Mode'** menu to **'Use Emitter Icon Color'**. Then the particles will all use the colour of the emitter icon.

In many cases the emitter will be in the hierarchy of an XP System object (which we will discuss later on). That too can have its own icon colour; you can force the emitter icon to be the same colour as the System object icon by clicking the **'Match System Object Color'** button. This will have no effect if the emitter isn't in a System object hierarchy.

6.1.2 Miscellaneous display options

There are a number of controls which let you alter what is displayed in the viewport. Most of these are self-explanatory and details can be found in full in the XP manual, but one or two need further explanation.

Force Display

This can be useful to speed up playback if you have a large number of particles which take time to draw. For example, suppose you have an emitter with two groups, one of which displays as spheres and the other as filled arrows. These are quite intensive when it comes to drawing them on screen, especially spheres, so it can be useful to turn the display to a less intensive one temporarily. To do this, you would first disable both groups in the emitter's Groups tab (so that they have a red cross icon) and then change the particle display menu to something like dots or squares. Now re-enable the groups. The particles will display as spheres or arrows until you enable the **'Force Display'** switch, and then they will display as dots/squares. This can really speed up playback in cases like this. Disabling the switch will revert the particle display to their actual shape.

Display Constraints

This switch will have no effect until you have some constraints in the scene, for example by adding a Constraints object. Then you can display the constraints as lines and can alter the colours by clicking the little arrow next to the **'Display Constraints'** label.

Filled circles

If you choose the 'Circle (Filled)' display type, there are several controls which only affect this mode of display. You can increase the number of segments - 8 is actually quite low and makes the 'circles' more of an octagon - plus give them an outline (whose colour you can change). These settings don't affect any other kind of display.

The other options in this section should all be self-explanatory.

6.1.2 Particle shape

The important thing to remember about shape is that it is purely a convenience. All particles are treated internally to XP as spheres. The radius of a particle is that of a sphere surrounding a particle. For the purposes of collision detection, the particle is assumed to be a sphere; this is true even if you are generating some geometry linked to a particle. The only time the shape becomes relevant in this sense is when using particles with xpBullet, and then the actual particle shape is the collision object - not just a sphere.

It is possible to render the viewport shape, but you need an extra object to do this. We'll look at the Display Render object in another chapter.

There are many different shapes to choose from the **'Editor Display'** menu, and in most cases it's just a matter of personal preference. The only thing to be aware of is that some shapes take longer to draw than others. This is especially true of spheres, which because they are shaded take a long time to draw. Filled circles are an alternative to spheres and are faster to display.

6.1.3 Particle colour

There are a number of colour options, some obvious and some less so. **'Single Color'** and **'Random Color'** mean what they say and don't require more explanation.

Use Shader

This mode gives you a shader link where you can add a shader or bitmap to colour the particles. Note that any particle can only have one colour, so this mode does not shade each particle with the complete bitmap or shader. What it does is sample the shader and colour the particle with the colour at the sample point.

But where is the sample point? There are two ways to do this. One is random: a point is generated randomly and the shader is sampled at that point. The more interesting one is shader space. With this, the particle position in the 3D world is used as the sample position. This can produce some interesting results. Consider this bitmap downloaded from Pixabay (<https://pixabay.com>) shown on the left, and on the right when we use that bitmap in the shader link with particle emission set to **'Regular'** mode (which gives us even spacing between particles) and **'Shader Sampling'** set to **'Random'**:



Figure 6.1. Bitmap used to colour particles

Which is hardly impressive but at least you can see what happens with random sampling. With the sampling set to **'Shader Space'** though, Figure 6.2 shows is the result. What we get is a slightly pixellated, almost pointillist effect - the image is now made up of small particles with colours sampled from the bitmap at a location calculated from the particle position. With even smaller particles we would approach the quality of the original image, but now we can move and manipulate the particles - almost as if the original pixels were being manipulated.

This file is in the download archive as [*cb6_emitter_particlecolour_shader.c4d*](#), including the bitmap used. Try this, then add a Turbulence modifier set to **'Curl'** mode with a low strength (about 1) for a nice effect.



Figure 6.2. Bitmap in 'Shader Space' mode

Gradient

There are two gradient modes. If you choose 'Gradient (Random)' you are presented with a gradient - which you can edit - and the particle colour is chosen randomly from the gradient. Much more interesting is 'Gradient (Parameter)' where the colour is chosen from the gradient using a particle parameter as an index into the gradient to select a colour.

With this mode, the emitter calculates a value from 0 to 1 from the parameter range supplied and the actual parameter value from each particle. This is then used as the index to fetch a colour from the gradient. If the value is zero, the colour from the left hand end of the gradient is used, if the value is one then the colour from the right hand end is used.

One possible difficulty is working out the range of values to use. As an example, say you have set the radius to 3 units with a variation of 2. Now you change the '**Gradient Parameter**' setting to '**Radius**' and by default the emitter will show minimum and maximum values of 0 and 100. However, the range of possible radius values is only between 1 and 5 (3 ± 2) so all the particles will have a colour from the left of the gradient. The result would look like this:

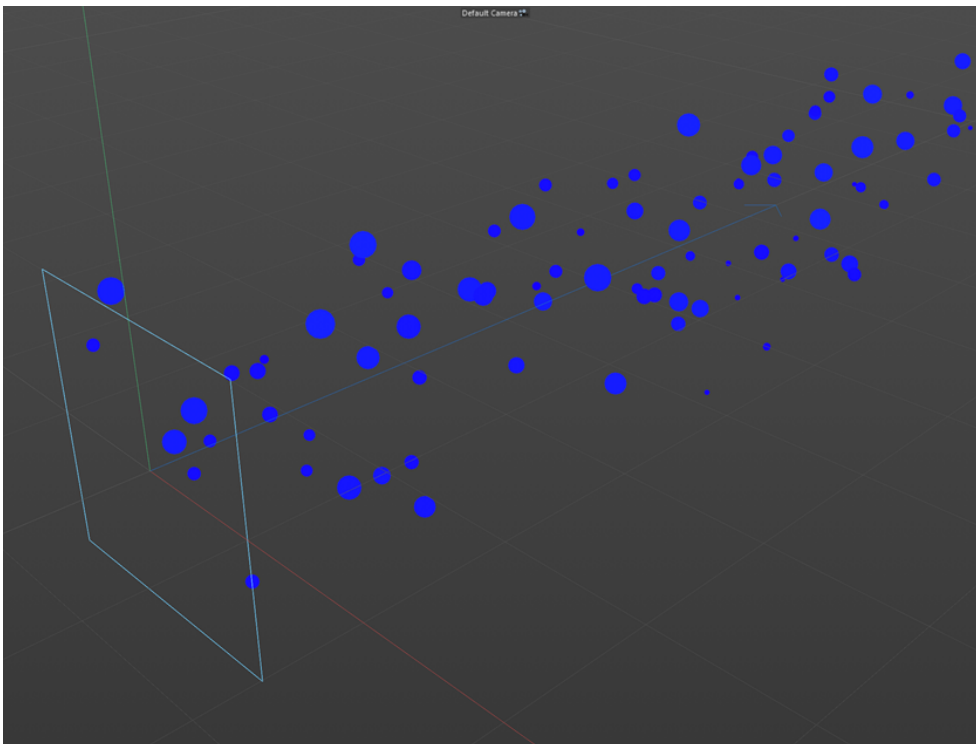


Figure 6.3. Using 'Gradient (Parameter)' mode to derive colour from particle radius

There are two ways to solve this. The first is manually to set the minimum value to 1 and the maximum to 5. But it isn't always that simple - you may not know what the minimum and maximum values will be - so an alternative solution is to enable the **'Auto'** switch. This will cause the emitter to work out the range from the minimum and maximum values of the radius of all particles each frame. Then the same scene would look like this, with the largest particles having the colour from the right hand end of the gradient:

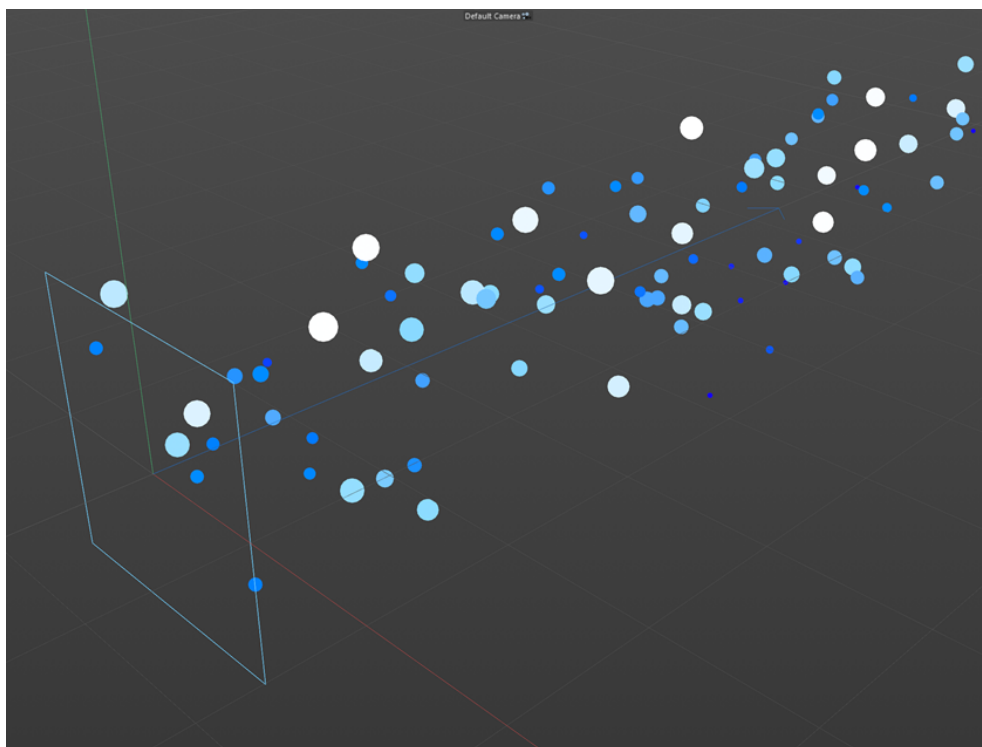


Figure 6.4. Corrected scene from Figure 6.3 using the 'Auto' switch

This scene file [*ch6_emitter_display_gradparam.c4d*](#) is present in the download archive for this chapter.

i Using the **'Auto'** switch solves another problem. As an example, if you have a Scale modifier in the scene which is increasing the particle radius, and **'Auto'** is disabled, eventually all the particles will have a radius which will give them the colour from the right of the gradient. If **'Auto'** is enabled, the relative range of radius values will be evaluated each frame and the particles will always have the correct colour from the gradient for their radius.

The **'Auto'** switch isn't available for all parameters. It's not available for age, and neither are the minimum and maximum settings, since they are automatically calculated from the scene length. Particle-particle distance has no auto setting because there is a large overhead in calculating this automatically each frame. Finally, direction and rotation are handled rather differently and don't have minimum and maximum settings at all.

If the parameter used is **'Direction'** the index into the gradient is calculated from the direction of the particle expressed in angles along the selected axis. That is, if **'Direction Axis'** is set to **'Heading'** the index value is zero if the heading angle is zero and 1 if the heading angle is 90 degrees (or -90 degrees). This is then used to get a colour from the gradient; if the particle heading changes, so will the colour. The same is true for the other two axes. There is also an option for **'HPB Combined'**. For this mode, the heading colour comes from the first third of the gradient, the pitch from the middle third and the bank from the final third. The three colours are then added together to make the final colour.

Using **'Rotation'** as the parameter is very similar except that the range for each rotation axis is set between 0 and 360 degrees. There is again a combined value and this time the three rotation values are simply added together.

⚠ There is a small but important caveat with 'Gradient (Parameter)' mode and particle groups. Suppose you have an emitter with a single group and the group colour mode is set to **'Single Color'**. Then you change the group, perhaps with a Change Group modifier, to a different group which uses 'Gradient (Parameter)' mode. You might expect that the particles would now adopt a colour dependent on whatever parameter was used in the second group, but in fact nothing happens. For technical reasons it isn't possible to change to a group using that colour mode. All other modes are fine, but not that one.

Object Color

With this option, if particles are emitted from an object the particle colour will be the object's display colour found in the **'Basic'** tab of the object. If the object's **'Display Color'** menu is set to **'Off'**, or if you choose this option in the emitter but are not emitting from an object, the project's default object colour, found in the project settings, is used.

6.2 The HUD

The X-Particles emitter can show some basic information in the viewport. If you enable **'Show HUD'** the emitter name and the number of live particles is shown at the top of the screen. If you need this but find it intrusive you can enable **'Only Show If Selected'** in which case it is only displayed if the emitter is selected in the object manager.

6.2.1 Particle information

You can display certain pieces of information against each particle if desired. To do this, turn on **'Show Particle ID'** which will display the particle ID number against the particle. Then from the **'Show Particle Data'** menu select the data item you'd like to be displayed. A variety of data is available including any custom data you may have added.

⚠ Drawing the text for particle data to the screen is very slow and if you have a lot of particles the frame rate will drop dramatically. If you want to see the data without this speed reduction, use the XP Console instead (we'll cover this in another chapter).

6.2.2 Text options

There are options to change colour and opacity but the most useful ones are the X and Y position settings, which will allow you to move the text to whatever position on the screen suits you best.

6.3 Modifier fields

Many modifiers in XP alter the particle velocity, which is a combination of speed and direction. The modifiers generate a vector - a line with both direction and magnitude (which gives the speed). These vectors can be visualised using this quicktab and this can be useful to show what the modifier is actually doing to affect particle motion.

Not all modifiers change particle velocity (speed and direction, remember) and even those that do may only do so under certain conditions. Those which do not will not show anything with this feature; the manual contains a list of the modifiers which do produce a velocity vector that will show in the display.

⚠ The modifier field is purely for visualisation and does not change the modifier operation in any way.

6.3.1 Vector colour and shape

Figure 6.5 is an example of the modifier field produced by a Turbulence modifier. For clarity, a [larger version](#) can be seen on the book's website.

There are a couple of points to note here. You can change the display from the default lines to dots (quick to draw but not very useful perhaps) or arrows, which are better at showing direction.

The colour of a vector indicates the change that this vector would make on the particle velocity, and this is also indicated by the line length or arrow size. If you want to see the direction only, and not show the effect of the speed change, enable the **'Direction Only'** switch. The lines/arrows will then appear all the same size.

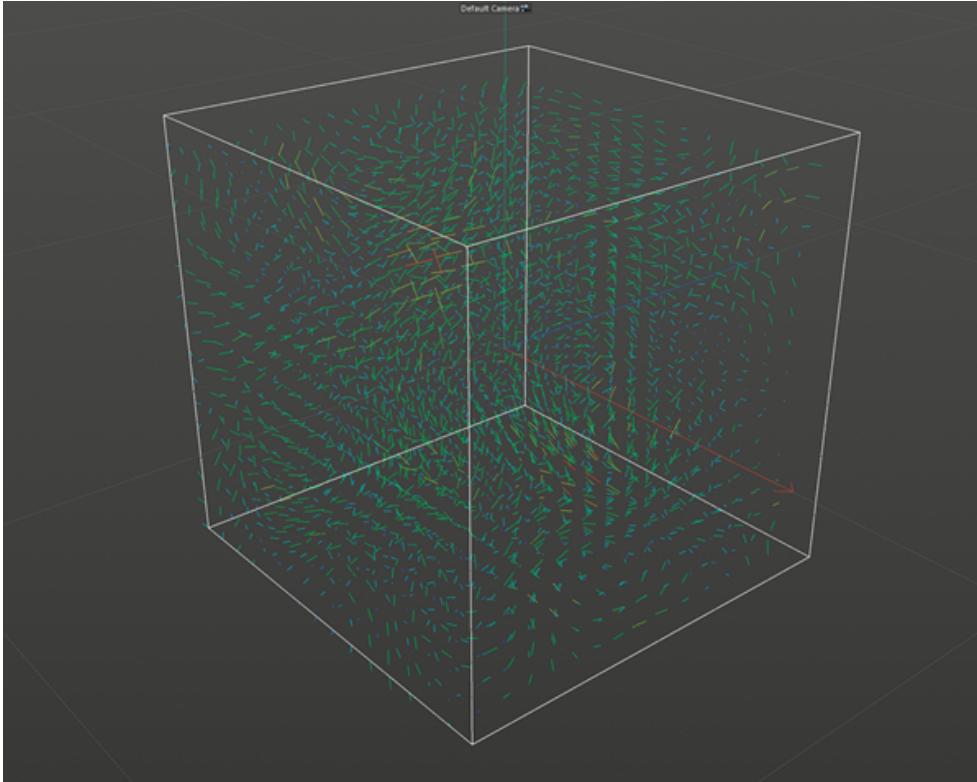


Figure 6.5. Turbulence modifier field display

6.3.2 Other options

One useful option is to change the resolution. Each vector is drawn inside a cell of the enclosing box; it is the cell size that varies when you change resolution, with the higher resolution having a smaller cell and therefore showing more vectors. Here is the field for a Turbulence modifier showing medium resolution on the left and high on the right:

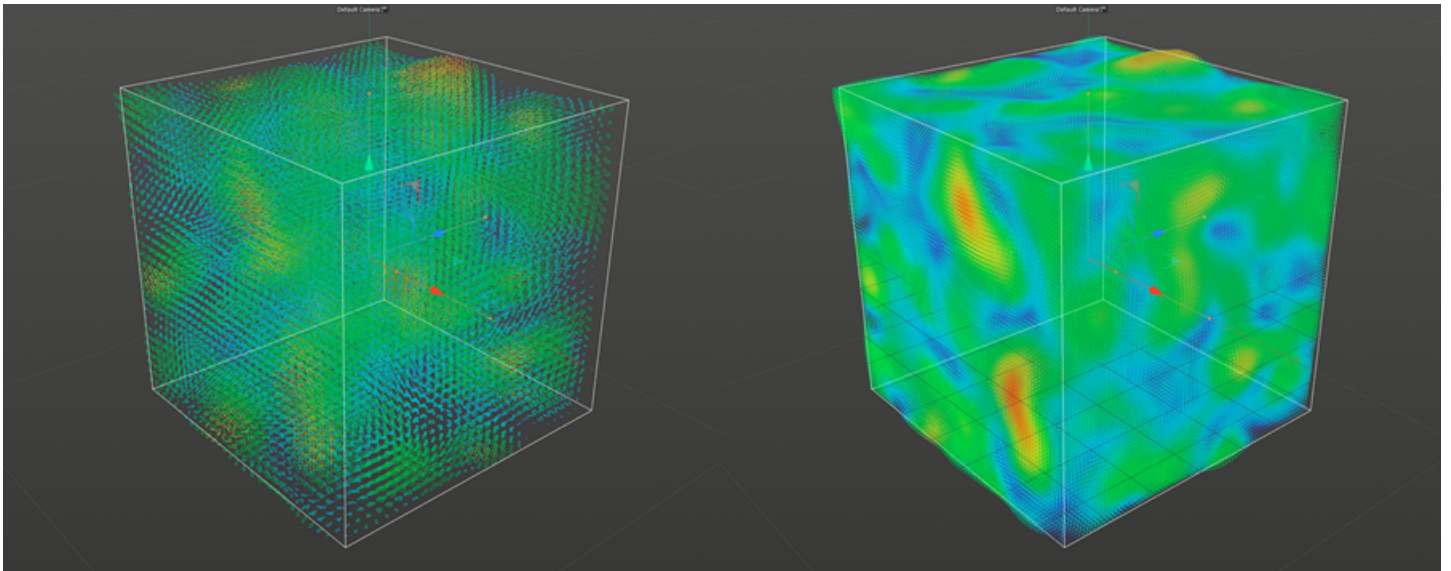


Figure 6.6. Modifier field display showing medium resolution (left) and high resolution (right)

You can also set a custom resolution and set the cell size manually to whatever you find best. Do be aware that if you set it too small, there will be so many vectors to show that it may take a very long time to draw them. You can also change the size of the enclosing box, but remember that this only affects the display and not the modifier function.

The display can be changed with the **'Color'** and **'Transparency'** fields, which need little explanation. But which colour is chosen from the gradient? This depends on the minimum and maximum speed changes from each vector. If the **'Auto'** switch is enabled, the emitter calculates these limits automatically, then chooses the colour from the left hand end of the gradient for the slowest speed

change, and from the right hand end for the highest speed change. You don't have to use the automatic setting, however. If you disable that switch you can set the minimum and maximum values yourself. As explained in the manual, these limits can be very small, so you may - depending on the modifier and the strength of its effects - have to reduce the maximum value to a small figure to see any difference (or even any vectors at all).

6.3.3 Include or exclude modifiers

What happens if you have two or more modifiers affecting the velocity? Their effects are combined to produce the final vector display. But you may not want that; you may want a modifier to be excluded from the display, or for only one of several modifiers to be used.

In this case, drag and drop the modifiers into the **'Modifiers'** list. In the latest version of X-Particles the **'Mode'** menu doesn't seem to have any effect. If you change the yellow tick icon against a modifier to a blue dash, the modifier won't be included in the display and only modifiers in the list with a yellow tick - or those not in the list at all - will be used.

6.3.4 Rendering the vectors

You can render these vectors using the Display Render object, which we will look at in a later chapter.

Chapter 7: The Emitter: Other Tabs

There are some other tabs in the emitter's interface still to be covered. These are as follows.

↓ The scene file(s) for this chapter can be found in the archive [examples_ch7.zip](#) on the book's website. Click here to [download the archive](#).

7.1 Groups tab

Particle groups were covered in Chapter 2. For an emitter to emit particles in a group, the group object(s) must be in the **'Groups to Use'** list in the emitter. Creating groups was covered in Chapter 2, but in summary you can create Group objects from the main X-Particles menu and drag and drop them into the list, or use the **'Create and Add Group'** button in this tab to do so automatically.

You can then set the distribution of particles into the various groups as described in Chapter 2. It is also possible to disable a group temporarily. If you have two groups in the list and change the green tick on one to a red cross, that group will no longer be used. If you disable all the groups in the list, the particle settings in the emitter will be used instead of those in the group(s).

7.2 Questions tab

This tab is used with the Question/Action system in XP. This will be discussed in full in Chapter 12 and Chapter 13.

7.3 Modifiers tab

By default any modifiers in the scene will work on all emitters. If you don't want that to happen, you can drag modifiers into the **'Objects'** list in this tab. If the modifier has a yellow tick icon, it will work on this emitter; if you click it to change it to a blue cross, it won't have an effect. You can use this so that a modifier will work on only the emitters you want it to and not on those you don't.

An alternative is to make the emitter part of the hierarchy of an XP System object. if you do that, then enable the switch **'Only in Same System'** only modifiers which are in the hierarchy of the same System object as the emitter will affect it.

7.3.1 Other objects

You might wonder why the list is labelled **'Objects'** and not **'Modifiers'**. This is because objects other than modifiers can also affect particles; you can use deformers and effectors to do so.

⚠ If you use a deformer to affect particles, you can choose to make the deformer a child object of the emitter or not. Normally, with other objects you must do that for them to work, but it isn't required with the XP emitter. However, if you do make the deformer a child of the emitter then the yellow tick/blue dash icons in the list won't work and the deformer will always affect the emitter. If you don't make the deformer a child of the emitter it must be added to the list to have any effect, but then you can turn it on and off with the tick/dash icons. Which method you choose depends on what you want to do and how much control you need over deformer effects.

Mograph effectors are treated in exactly the same way as deformers.

7.4 Editing tab

Once particles have been generated you can edit them using this tab. When you enable editing, internally X-Particles creates a hidden point object with no polygons but one point per particle. You can then use Cinema's editing tools to move the particles around and (in R19 and earlier) delete unwanted ones.

7.4.1 Editing mode

Points

In **'Points'** mode the procedure is very simple. In the editing tab, turn on **'Editable'** and play the animation to the point at which you want to start editing. At this point, if you change to Cinema's points mode the particles will disappear and you see a point cloud instead. You can either edit the points in this mode or leave the particles visible and edit them directly. Changing to points mode lets

you do things such as select points and create a selection from them, which you can't do with particles. If you need to select some particles, you must be in points mode to do it.

Now you can manipulate the particles or points however you need to. If you edit in points mode, when you change back to object mode you will see that the particles have been updated to match the points.

! Unfortunately one thing you can't do is delete the particles or points if you are using Cinema 4D R20 or higher. Maxon made a very significant change to the SDK for that version which removed the method used for deletion, and it hasn't been possible yet to find a workaround.

Note that whenever you edit the particles a keyframe is added. This will show as 'X-Particles Edit Data' in the timeline window.

Once the edits are complete, rewind the scene and play it back. You will notice that the changes you made do not occur instantaneously on the frame you made them but start at some point before the change was made. This is the effect of the **'Range'** setting which gives the number of frames the changes will start at before the frame on which the edit was actually made.

If you look at the scene file [cb7_emitter_editing_points.c4d](#) in the download for this chapter, you can see that the edits were made on frame 60 (that's where the keyframe is) but the particles start to move to those changes at frame 40 (because the range is 20 frames and $60 - 20 = 40$). The movement is also controlled by the **'Blend'** spline which by default gives a smooth transition to the final edited change. Try experimenting with the range and the spline to see the different results.

Paths

The other mode lets you manipulate the path a particle has taken to its current position. With this mode selected, and Cinema 4D in points mode, when you play the animation you see this (one particle shown for clarity):

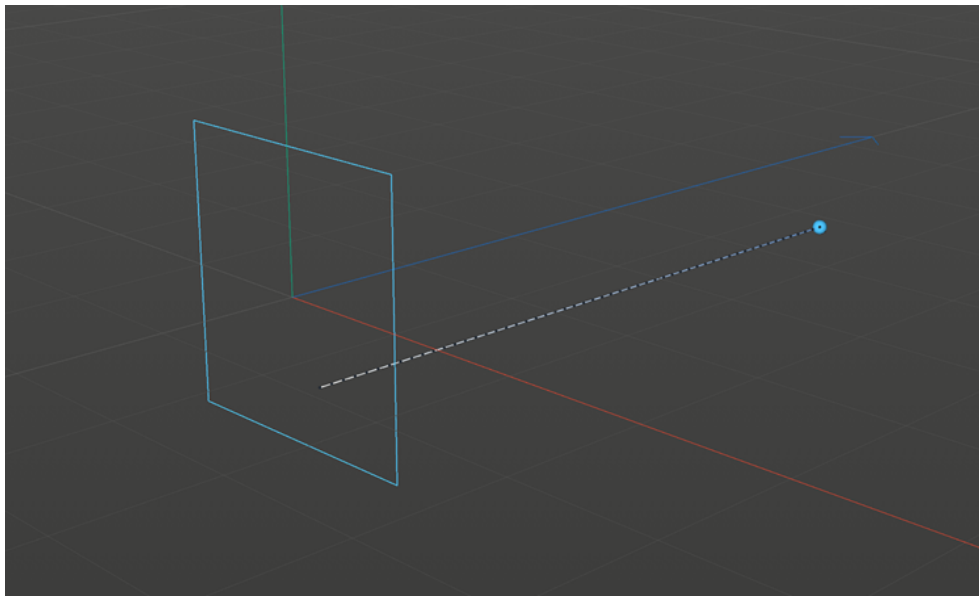


Figure 7.1. Editing a particle path

The path of the particle shows as a spline with one point per frame. You can then edit the path as required and on playback see something similar to Figure 7.2. Again, you can change the **'Range'** value and **'Blend'** spline as required.

7.4.2 Removing editing data

To remove edits made on a particular frame, go to that frame in the timeline and click **'Clear Frame'**. To remove all editing data simply click **'Clear All'**.

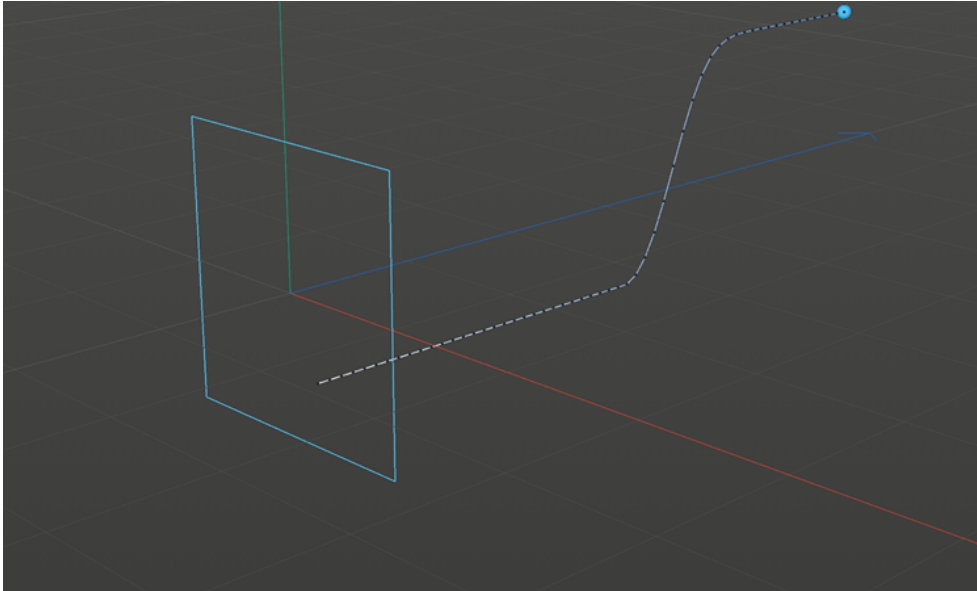


Figure 7.2. Result of editing a particle path

7.4.3 Blending between multiple edits

If you edit the particles on more than one frame the emitter must blend between the various edits. The **'Blend Mode'** menu controls this and has two options. The **'Clipped'** option blends between an edited frame and the next edited frame, while **'Mixed'** mixes the **'Range'** values for the edited frames. It's difficult to describe this and in many cases you will see little difference between the two, but it is worth experimenting to see the effect of the different modes.

7.4.4 Editing painted particles

You can paint particles on the surface of an object using the Particle Paint tool. This is discussed in a later chapter and this section will be covered then.

7.4.5 Deleted particles

⚠ This section only applies to XP when used in C4D R19 or earlier.

If particles were deleted by editing them, you can make them visible again (but they are still deleted) by enabling the **'Show Deleted'** switch. Having done this you can use the Particle Paint tool to restore the deleted particles.

To restore all the deleted particles you can click the **'Clear Deleted'** button.

7.5 Advanced tab

This tab only has two elements.

Threading

X-Particles is fully threaded for maximum speed. Normally you should leave this switch enabled, but if you need to turn threading off, disable it. Note that this will slow the whole of XP significantly.

Random Seed

Many aspects of XP use a random number somewhere and although some have their own seeds, many, especially the modifiers, use this seed value. Whenever you create an emitter it is created with a random seed value, so the particle generation and modifier effects should be different for different emitters. If for some reason you need to emitters to show the same pattern of particles, make their seed values identical.

Chapter 8: Rendering particles

Once particles have been generated, you might want to render them. There are several ways to do this:

1. With the Cinema 4D standard renderer, use the X-Particles material
2. Use a third-party render engine such as Cycles 4D (which is optimised for use with XP) or another engine which can render the particles directly
3. Use the XP Display Render object
4. Or render them indirectly by attaching simple geometry to them using the XP Generator or Sprite objects

Which one to use is a matter of preference to some extent. However, if you want to render the actual particle shape such as the arrow or pyramid shape, you need the Display Render object. And of course, attaching geometry is cheating! In this chapter we will look at the first two options.

↓ The scene file(s) for this chapter can be found in the archive [examples_ch8.zip](#) on the book's website. Click here to [download the archive](#).

8.1 The X-Particles material

This was developed before many of the current third-party render engines existed or supported XP. It still has its uses; some of the screenshots in this book were made using it for sheer convenience. Generally however, you will get better results with Cycles 4D or another render engine.

8.1.1 Using the XP material

Not all the options will be covered here, just those to get started so you can render particles without geometry. To use it, create the XP material from the material manager (it's in the 'Extensions' sub-menu of the 'Create' menu of the manager in the latest versions of C4D) and apply it to the emitter. That's all you need to do, then just render the scene.

Note the the particles are rendered as unshaded discs by default but you can change the shading in the '**Illumination**' tab. The shape is always a disc, however, regardless of the shape in the viewport. Here is a basic render with the colour set to random in the material itself:

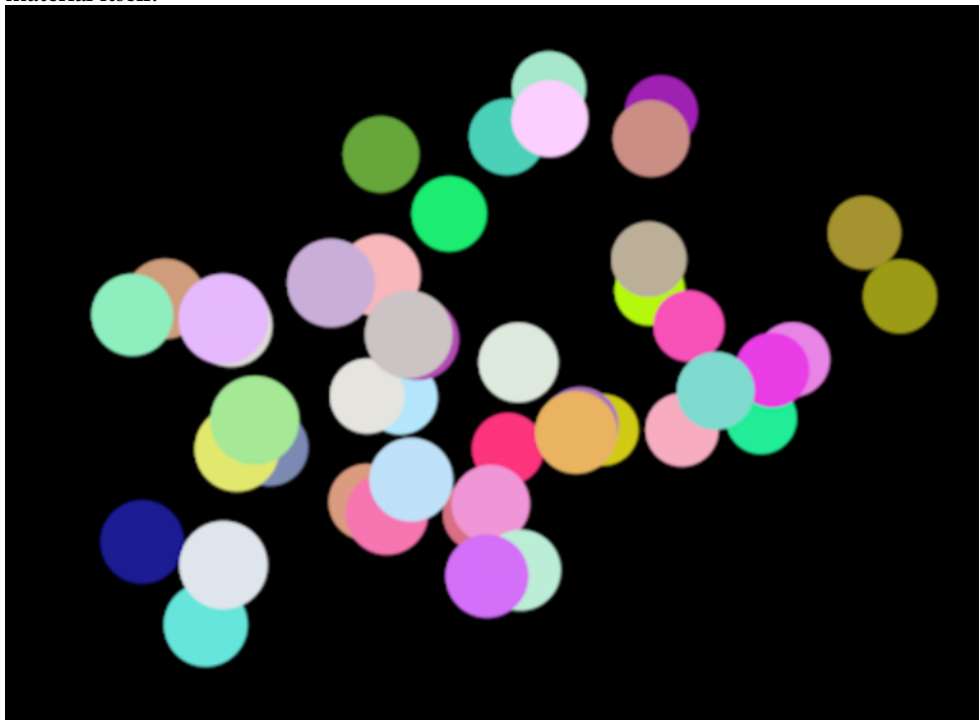


Figure 8.1. Using the XP Material to render particles

You can also apply the material to a particle group, and if you do that, only the particles in that group will be rendered. That means you can render different groups with different material instances if desired.

8.1.2 Colour options

There are numerous colour options. You can use the particle colour, which is the obvious choice and the default option. Or you can use a single colour, a completely random colour for each particle, or a colour randomly selected from a colour gradient. In those last three cases, the rendered colour will be different from the particle colour in the editor.

On top of the colour you can choose a shader or bitmap. If you do that, the particle will be rendered with the complete shader or bitmap, unlike the same option in the emitter Display tab, where the particle colour is sampled at just one point from the shader/bitmap. The shader can then be blended with the selected colour. In this screenshot, the colour was set to random and a checkerboard texture blended with the coloured particles:

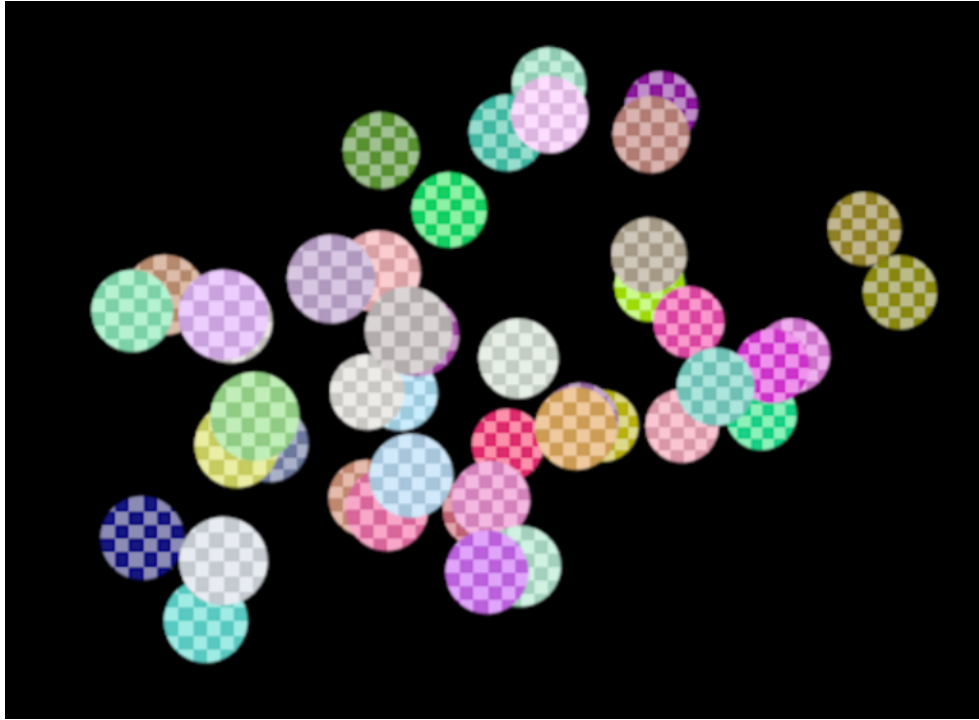


Figure 8.2. Adding a checkerboard shader to the colour in the XP Material

8.1.3 Colour modifiers

The Color tab of the material also has a section labelled **'Modifiers'**. These are not the same as the particle modifiers found in X-Particles. They act to modify the colour chosen but don't affect anything else. In Figure 8.3, the **'Distance'** modifier is used, which selects a colour from a gradient according to how far it has travelled since being created (you can and should adjust the distance range to get this looking how you want it).

There are a number of different modifiers and you can combine their output with each other and with the main colour using the **'Blend'** menu and **'Mix'** slider.

8.1.4 Transparency and Size

As the names suggest, you can use these tabs to control the transparency and size of the rendered discs. They also have modifiers which are used in the same way as for colour.

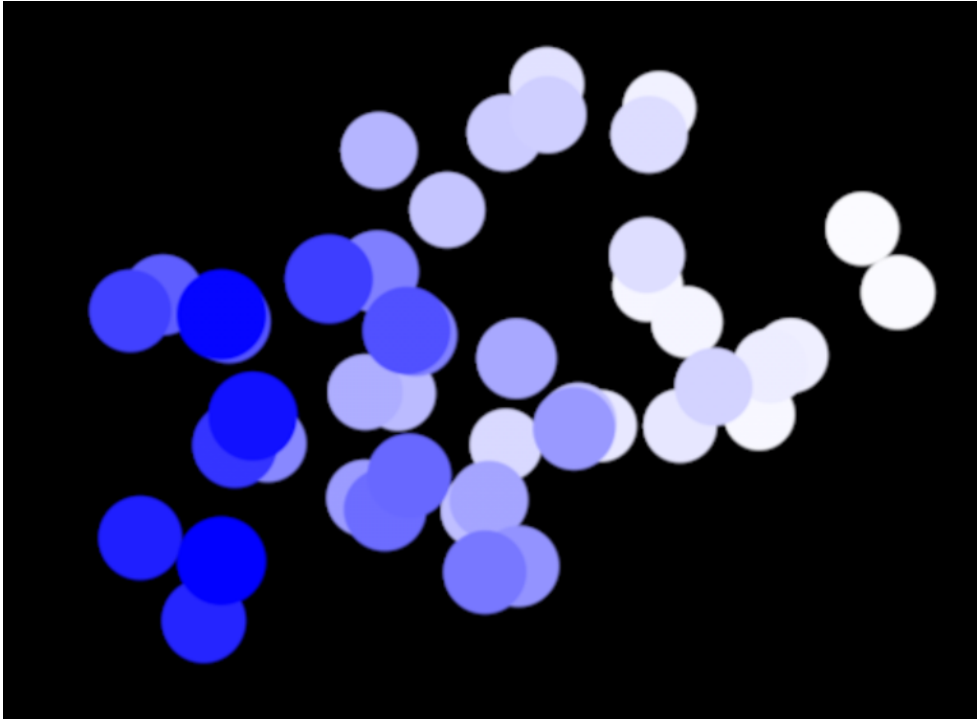


Figure 8.3. Using a colour modifier ('Distance') in the XP Material

8.1.5 Filling

This is an interesting tab. If you enable the **'Fill'** switch then at render time additional particles are rendered which can help to fill in gaps between the actual particles. Bear in mind that these are only present at render time and do not affect the simulation at all - they are not in the scene during playback. In this example, the number has been reduced to two from the default 10 and if you compare it with the first screenshot in this section you can see that each 'fill' particle copies the size and colour of the original it is associated with:

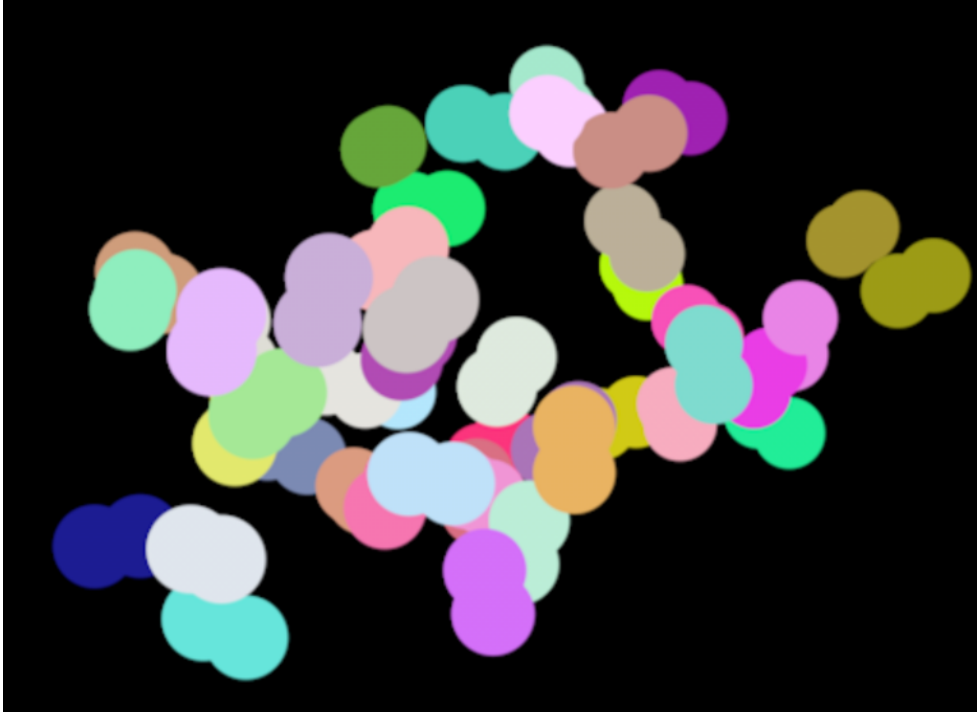


Figure 8.4. Using 'Fill' in the XP Material

It is possible to add colour, transparency and size variation to the fill particles to give a more varied appearance.

⚠ The thing to remember with filling is that the **'Radius'** parameter is NOT the radius of the fill particles but the radius of a

sphere around each original particle. It is in that virtual sphere that the fill particles are rendered.

8.1.6 Volumetric

This rendering method has been pretty much superseded by using other objects such as Explosia FX and volumetric rendering in Cycles 4D and other render engines. You can use it to obtain very simple (and not very realistic smoke effects) without the use of more complex but much better methods.

8.1.7 Illumination

The default illumination model for the rendered particles is flat (non-shaded). But there are several others. Here are examples of the more useful ones:

Flat:

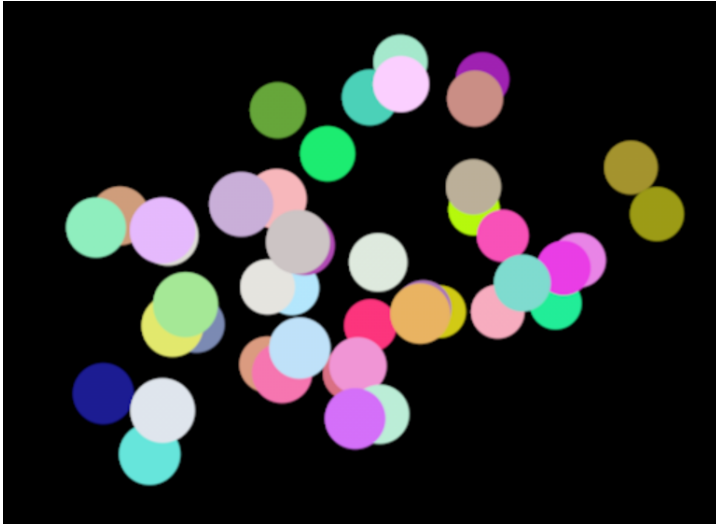


Figure 8.5. 'Flat' illumination

Diffuse (similar to Flat, but shaded):

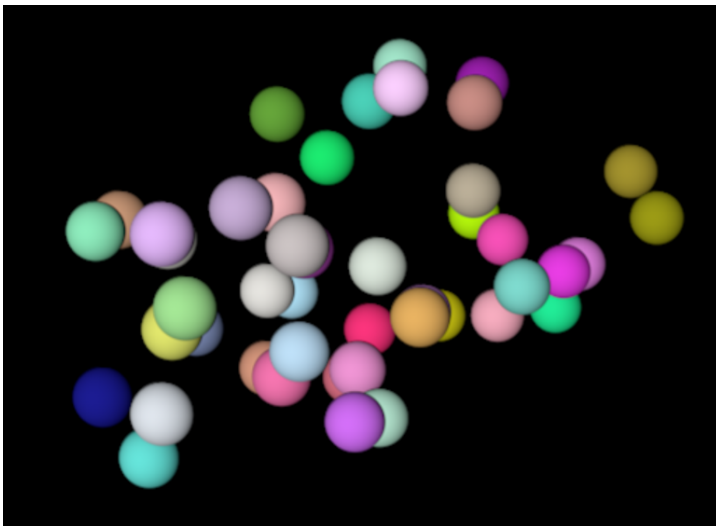


Figure 8.6. 'Diffuse' illumination

Phong (similar to Diffuse but with a specular highlight):

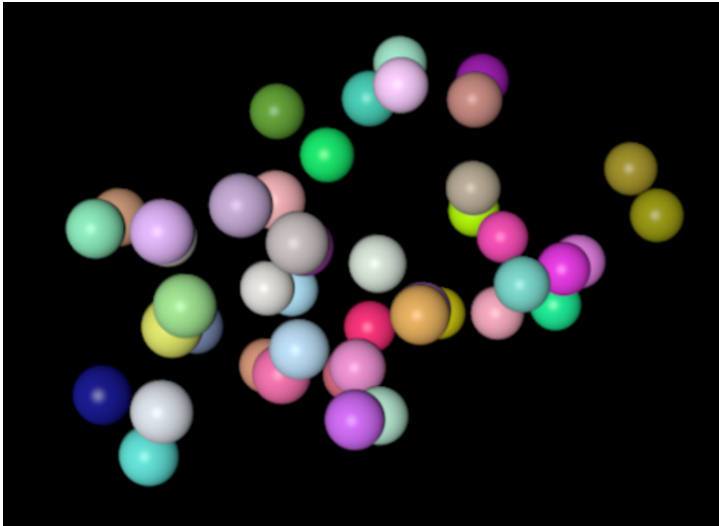


Figure 8.7. 'Phong' illumination

Fuzzy:

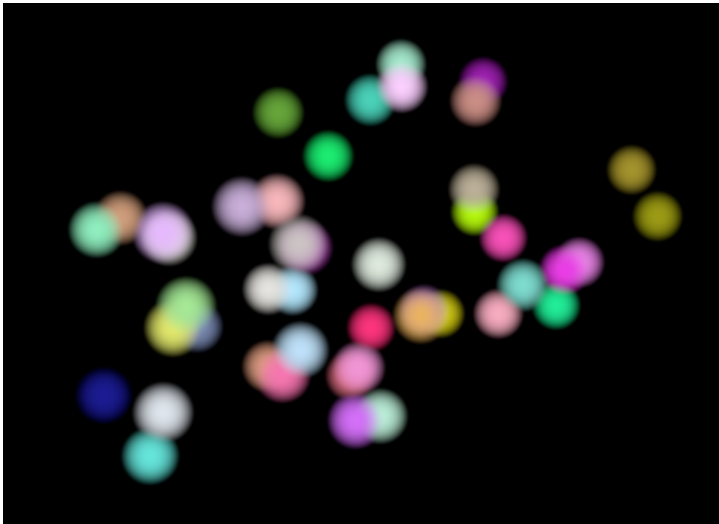


Figure 8.8. 'Fuzzy' illumination

Neon:



Figure 8.9. 'Neon' illumination

8.1.8 Spline rendering

Although the XP material was developed for rendering particles, it can do one other thing that can be quite useful: it can render splines without geometry. Here is a Star spline rendered just with the material, using Phong illumination. There is no geometry here at all, just the spline with the material applied:

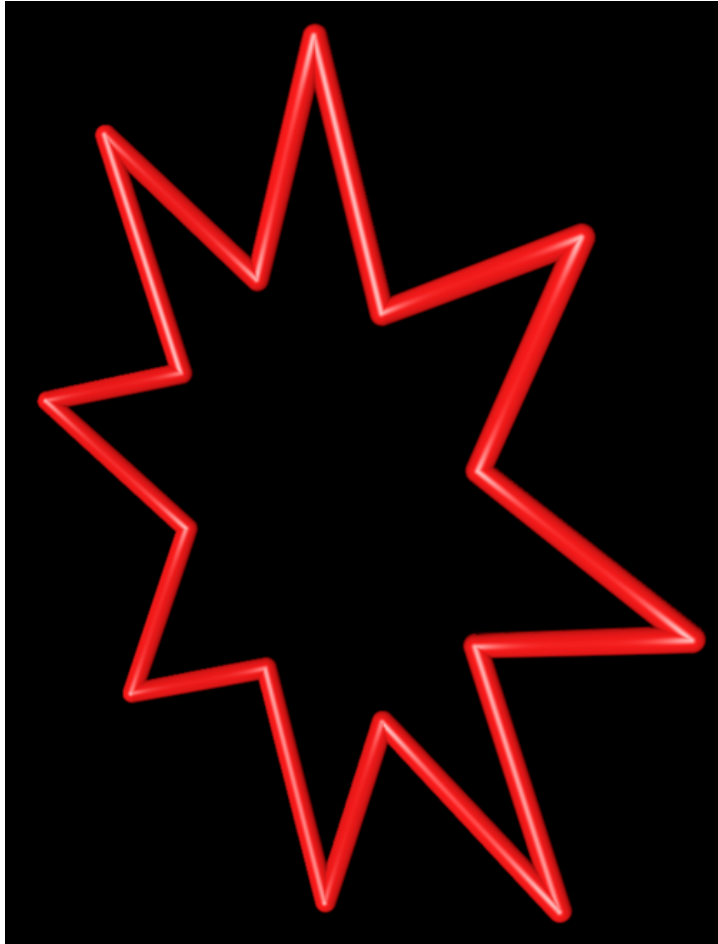


Figure 8.10. Spline rendering with the XP Material

8.2 Rendering particles with Cycles 4D

If you have Cycles 4D, which you will if you have the new 'Insydium Fused' edition, then you can render particles with that render engine. Setting this up is simple. Create a new emitter, then create a Cycles 4D 'X-Particles Color Emission' material and drop the material onto the emitter. Then switch to using Cycles 4D in the render settings and the particles are rendered as self-illuminating flat discs.

An example file is provided as [ch8_rendering_cycles4d.c4d](#) in this chapter's download archive.

i Note that the particles will emit light using this method. You can change the Emission shader in the material to a Diffuse BSDF shader and then no light is emitted, and the particles are rendered as shaded spheres. To see them, however, you will need some light in the scene. The example file contains this alternative - in the material, change the shader used to the Diffuse BSDF and turn on the Cycles 4D light in the object manager.

Summary

The XP Material provides a way to render particles without adding geometry, but is now outmoded compared to third-party render engines. As a means of rendering particles and splines it still has a certain utility but for best results a different render engine is recommended.

(This page intentionally left blank)

Part 2: The Middle Ground

In this part we will be looking at some more advanced features of X-Particles, such as the generator objects, the modifiers, and techniques to do with questions and actions and data handling.

Generators Part I

X-Particles has a number of objects under the heading 'Generators'. These are objects which as the name suggests actually 'generate' something. In some cases this will mean particles but usually it implies some kind of geometry. We'll start with the 'big three' generators - the Generator object itself, the Trail object and the Sprite object.

Chapter 9: The Generator Object

The Generator object is used when you want to 'generate' a piece of geometry for each particle. Each generated object is linked to a specific particle. The actual object generated will be a copy of the original but some aspects of it - such as scale, colour and so on - will be set from properties of the particle. More importantly, when the particle moves the generated object will move too, and when the particle dies, the generated object will be removed from the scene.

The term 'generator' in Cinema 4D means that an object is being produced parametrically rather than with points and polygons. Many objects in Cinema are generators; as well as the Cloner, the Extrude, Sweep, Subdivision Surface, and many others are generators, including primitive objects such as the Cube, Sphere, Circle spline, and so on.

If you are used to using the Mograph cloner to generate copies of an object, the Generator is very similar but the advantage is being able to move the object along with its associated particle. Since you can use X-Particles modifiers to affect the particle, the generated object will be affected accordingly - so you can change the colour, size, spin, speed and direction of movement and so on of the generated object.

You can generate many different objects using the Generator, including objects generated by other Cinema 4D generators. For example, you can make a Mograph Cloner a child of the Generator and for each particle a copy of the Cloner and its cloned objects will be generated. There are some things though which you can't generate. You can't generate an XP emitter from an XP Generator; if you try, it will look as though it works but the generated 'emitters' are just a drawn shape and won't emit any particles. There may be other objects that the Generator can't reproduce, which in some cases may be due to the priority system in C4D.

↓ The scene file(s) for this chapter can be found in the archive [examples_ch9.zip](#) on the book's website. Click here to [download the archive](#).

9.1 Generating objects

Generating a single object for each particle is the simplest possible use of the Generator. To generate an object, simply make it a child object of the Generator and drag an emitter into the '**Emitter**' link field in the Generator interface. When the animation plays, an object will be generated for each particle the emitter produces.

9.2 Generating multiple objects

It's simple to generate one object for each particle, but what if you want to generate more than one object? There are three ways to do this.

9.2.1 Generate different objects for each particle

Suppose you have a scene with a Generator with two child objects - a sphere and a cube. When the animation plays, each particle will either be linked to a sphere or a cube. By default this is done sequentially, so if the first child object is the cube, the first particle will be associated with a cube, the second particle with a sphere, the third with a cube again, and so on.

You can change this by using the '**Use Multiple Child Objects**' menu. This is covered thoroughly in the XP user manual so won't be explained again here. But whichever method is chosen, each particle will still be linked to only one object.

9.2.2 Put the objects to be generated into a Null object

If you make the objects you want to generate child objects of a Null object, you can then make the Null object the child of the generator. In this case, each particle will be linked to an instance of the Null object so the two (or more) objects that are child objects of the Null object will be generated for each particle.

9.2.3 Use multiple Generators

You can attach more than one Generator (generating different objects) to the same emitter. This isn't normally recommended since the objects from different Generators will be generated at the same position, rotation, etc. but you can vary some parameters. For example, if you have two generators you can set the scale for one within the Generator itself, but leave the other to derive its scale from the particle. This might be useful in some cases.

9.3 Generating animated objects

If you have an object which is animated using point-level animation (PLA) the animation will be reproduced in the generated objects. The animation will start from the same point for each object, which might look odd in some scenes. Take a look at the example file [ch9_generator_anim_bird.c4d](#). This has a (very) stylised bird-like object flapping its wings. If you play the animation, note that all the 'birds' flap their wings in unison - very nice formation flying but not what you would expect real birds to do!

You can specify an offset from the start frame but the animation will still look the same for all objects. To fix this, you can tick the **'Random Offset'** switch in the Generator. Now each animation will start at a random frame so the birds don't all flap wings the same.

⚠ There is an important point to note here - the animation will not loop. If you have a 90-frame scene like the above example, for each bird the animation could start at anywhere from frame 0 to 90. That means that after 90 frames the animation will stop and the birds won't flap at all. For example, if the object's animation started at an offset of 80 frames, it will stop animating when it reaches frame 10. The only way around this is to make the animation longer than the scene. In the example file, the scene is 90 frames long but the animation is 180 frames long, so the birds never stop flapping.

In addition to offsetting the start frame, you can change the 'scale' of the animation. This is actually the speed with which it is carried out. In the example file, try setting the **'Scale'** and **'Variation'** both to 50%. You will see that the birds' wing now flap at different speeds.

9.4 Other settings

9.4.1 Offset

By default the generated object will be located at the particle position - specifically, the axis of the object will be located at the particle position. If necessary you can offset the object from the particle using the **'Offset'** setting, and add variation to that with its **'Variation'** parameter.

If the axis is at the geometric centre of the object, which is often the case, the object will be centred around the particle; but that is sometimes undesirable. Consider the scene in Figure 9.1 in which the small blue cones are being emitted using the large pink cone as the source.

You can see that the generated cones appear to be partly embedded in the source object. You might prefer that the base of the generated cone was resting on the source object surface. This can be done by ticking the **'Offset to Base'** switch and then you would see the corrected result in Figure 9.2. Now the generated cones rest on the surface of the large cone. This file is included in the examples for this chapter as [ch9_generator_offsettobase.c4d](#).

⚠ This only works correctly if the axis of the object to be generated is at the centre of the object's bounding box. This is the case for many objects but certainly not all.

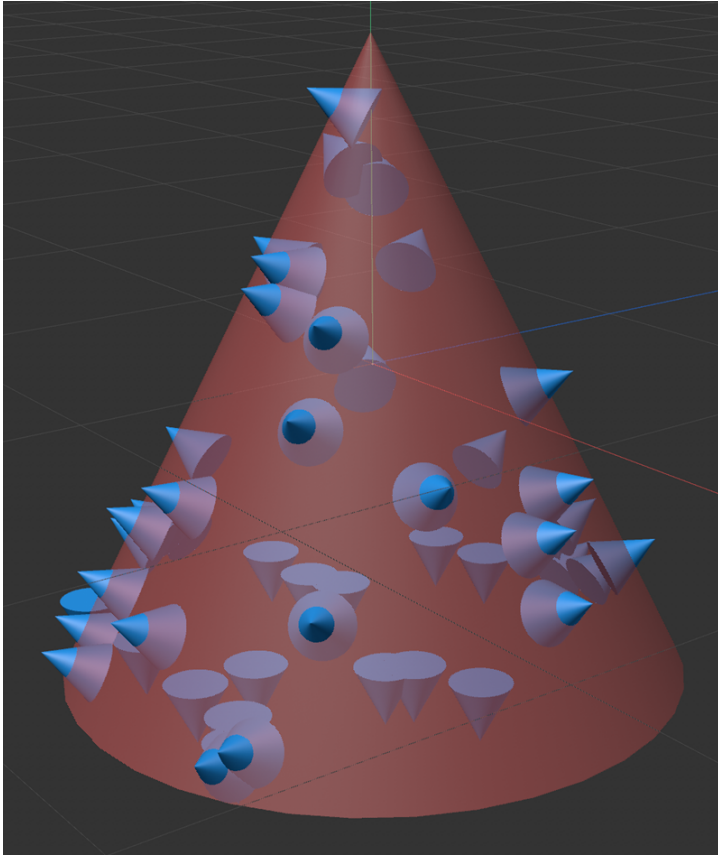


Figure 9.1. Generated objects partially embedded in the source object

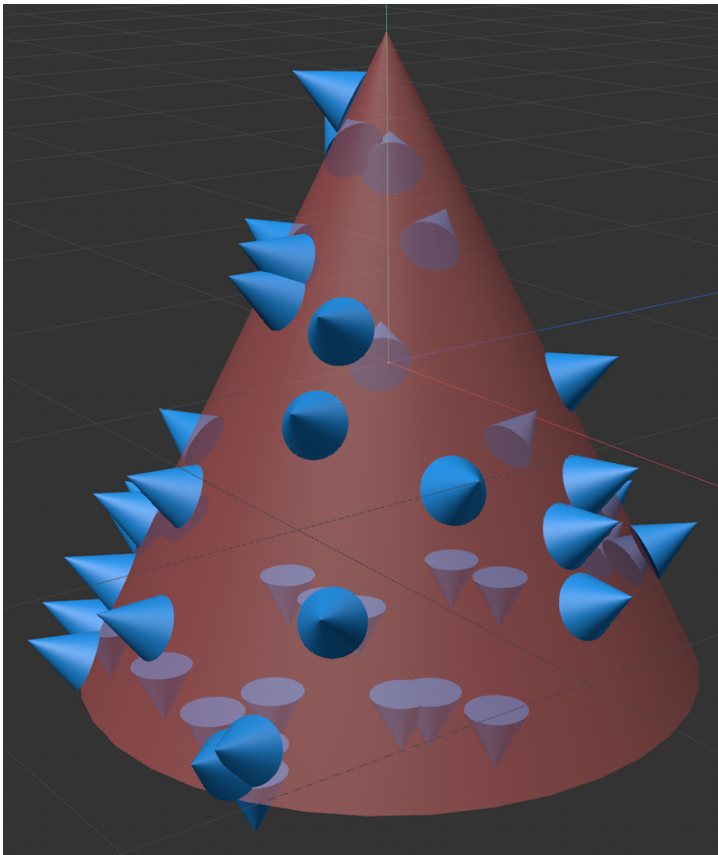


Figure 9.2. Same scene as Figure 9.1 with 'Offset to Base' switch used to correct the embedding

9.4.2 Scale

The size of the generated objects depends, by default, on the particle radius. Since the radius can vary between particles, how does the Generator vary the size of the object?

It does this by changing the scale of the generated objects. The radius of the particle is taken and the scale of the object is calculated by dividing the particle radius by 100. This means that if the particle radius is 100 scene units, the generated object scale will be set to 1 on each axis. You can see from this that if the particle radius is small, as it usually is, the object scale will also be small. You can see this if you generate some objects then invoke 'Current State to Object' on the Generator and look at the scale of the generated objects.

However, you aren't restricted to using the particle radius to determine the object size. With the '**Scale Using**' menu you can set the object scale from the particle scale - remember that this is 1.0 on all axes by default, so you will generate objects as large as the original source object. Or, you can use the source object's own scale (again, this is 1.0 by default) so that you could, for example, animate the source object scale and the generated objects will change size as the animation runs. You can even set the scale directly in the Generator, with variation per-object if desired, and animate that. This all gives you considerable flexibility in controlling the size of the generated objects. And of course, you can use the Scale modifier to change the particle radius or scale and change the size of the generated objects accordingly.

9.4.3 Rotation

Rotation works in pretty much the same way as scale - you can use the particle's rotation to rotate the objects, use the source object's rotation or set the rotation directly. There is one other option - '**Offset Particle Rotation**'. This lets you enter a rotation (with variation if required) and this is then added to the particle's own rotation to give the final rotation applied to the object. Most of the time though, you will probably want to use the particle's rotation which you can control using a Spin modifier.

9.4.4 Display

This is a separate tab and lets you set display options which are a subset of those found in the Cinema 4D Display tag. You could do exactly the same by adding a Display tag to the Generator object, so this is simply for convenience and doesn't add any new features.

9.5 Object morphing

Morphing was a frequently requested feature in X-Particles but doesn't seem to be used often. Since it requires a Generator object to make it work, we'll cover it here rather than in a separate section.

9.5.1 Overview

The purpose of object morphing is to change the shape of an object when the animation runs. You aren't restricted to morphing to one other object and that's it; you can morph a morphed object into another one and repeat as often as desired.

The main catch with the method used in XP is that the initial object and the morphed object(s) must have the same point count. If they don't, it simply won't work. This does restrict the nature of the final morphed shape; it would be very difficult to morph a sphere into a cube, for example, but not impossible.

To morph a generated object you will need:

- the initial generated object
- one or more new morph targets
- one or more Morph modifiers

The Generator itself morphs the object, which it does by moving the vertices to a point between the initial object the morph target.

9.5.2 Simple morphing

For the simplest possible example, see file [ch9_generator_simplemorph.c4d](#). This shows a pyramid morphing into a cube. It works like this:

- the Generator has two child objects - the original one (the pyramid) and the object to morph into (the cube)
- it is set to generate only the first child object (which can be set from the **'Multiple Child Objects'** menu)
- it has one morph target (the cube) in the **'Morph Targets'** list
- the Morph modifier has the default settings - there's no need to change anything

When the animation plays the object will morph into its new shape over a period of 20 frames (5% change per frame takes 20 frames to complete).

9.5.3 The Morph modifier

There are a couple of other points to make about the modifier settings. The **'Operation'** can be changed to **'Set Value'** in which case the object will morph instantly to the target shape; you can also change the **'Morph Max'** setting, so that the morphing will stop when it reaches the amount specified in that setting. Most of the time you will probably want to leave these settings at the default.

'Rate of Change' is self-explanatory but what about **'Change Morph Target'** and **'Morph Target Index'**? If **'Change Morph Target'** is turned off, which it is by default, the modifier will always use the first morph target in the list - that is, it will morph the generated object to the first morph target. But you might want to use the second target in the list, if there is one. In that case, you would turn this switch on and set **'Morph Target Index'** to 2. As an example, suppose your emitter is emitting particles in two different particle groups. You could use two Morph modifiers and have one morph particles in group 1 into the first target, and the other morph particles in group 2 into the second target. We'll look at a more complex example shortly.

The **'Threshold'** setting is only relevant if the modifier uses a falloff (pre-R20) or field to control the effect. Without one, the modifier will always work but if there is a field/falloff, it will only work if the particle is within the field/falloff's zone of effect. The **'Threshold'** setting determines how strong the field effect must be before the modifier actually works - it doesn't control the 'strength' of the modifier. For example, with a spherical falloff, the zone of effect is at 100% only when a particle is within the inner red sphere; the same is seen with a spherical field. If the **'Threshold'** setting is left at 100%, this means the modifier will only work when the particle enters that inner sphere, but if it is lower than 100%, the modifier will also work somewhere (exactly where depends on the threshold value) between the outer and inner spheres.

Finally, there is the switch labelled **'Switch Generator to Morph Target'**. The manual explains this by stating that, if you have a complex object with a lot of vertices, if this switch is off then the generator will have to morph the object every frame, even when the object has fully morphed, and that this could take some time.

But there is a limitation. Imagine that you have an object with 1000 vertices and you generate 1000 of them. Each frame, the Generator has to iterate through each object, moving each vertex to a new position. That will take time, slowing down the playback. Now suppose that the morph has reached 100%, that is, the object is fully morphed; a lot of time could be saved if the Generator generated the morph target object directly, so it didn't need to do any morphing at all. Turning on the **'Switch Generator to Morph Target'** switch will do that, but you can see the problem: if the maximum morph is anything less than 100%, the object will snap to the fully morphed version as soon as it reaches the maximum morph value.

This switch has a particular application when you want to perform morphs in sequence, as discussed next.

9.5.4 Sequential morphs

One thing you might want to do is have an object morph to one target, then have the morphed object morph to another target - and perhaps even longer chains. You can do this but it requires a bit more work and involves using Questions and Actions.

The main problem is that a morph modifier can only control morphing between one object and another; in its present form it cannot deal with sequential (chained) morphs. To do that, we will need more than one modifier, but then how do we prevent all the modifiers trying to morph the object at the same time?

The answer is to use the modifiers in action-controlled mode. The advantage is that each modifier can be turned on and off for individual particles, so we can control which modifier will work on a particle at any stage. In the next example we want the Generator to generate a pyramid shape which morphs into a cube then back into a pyramid which is a reverse of the original. If you load the example file [ch9_generator_twomorph_base.c4d](#) we'll set this up so you can see how it works. (If you just want to see the completed file without having to set it up yourself, you can load the working file [ch9_generator_twomorph_final.c4d](#).)

There are three objects to be generated - the original pyramid, then the cube, and then the reversed pyramid. All three are child objects of the Generator which is of course set to generate only the first child object - the initial pyramid. There are two morph targets

in the list, the cube and reversed pyramid. This is all the same as the simple morph so far. Now add a Morph modifier to this scene. If we don't change any settings, it will morph the pyramid into the cube and that's all. What we want to happen is that, when the pyramid is fully morphed into the cube, the modifier should stop acting on the particle so that another modifier can step in and carry out the second morph to the reversed pyramid. Because we are going to add a second Morph modifier, we should rename the first one to avoid confusion. I'd suggest renaming it to 'Morph to Target 1' but as long as you can distinguish the two modifiers it doesn't matter what the name is.

Next, change the **'Mode'** setting in the Morph modifier to **'Action Controlled'**. Now if you play the scene, nothing happens at all. This is because in action-controlled mode, a modifier has to be instructed to start working on a particle. In this case, we'll use a Question object to do that. In the emitter, go to the Questions tab and click the **'Add Question'** button. This will add a Question object to the scene, which will be a particle age question. That's fine, we can use that question type, but we need to change the mode to **'Equals'** and then set an age when we want the modifier to start working. A value of 10 frames would be fine, but you can set it to anything you like. Finally, drag and drop the Morph modifier into the **'Modifiers to Activate'** list. What this will do is test the particle age, and when it reaches 10 frames, will turn on the modifier for that particle.

Playing the scene will show that the morph starts at frame 10 and takes 20 frames to complete. At this point we need to make an important change to the modifier's settings, which is to turn on the **'Switch Generator to Morph Target'** switch. This forces the Generator to generate the morph target (i.e. the cube) directly rather than generating a pyramid then morphing it to a cube. But it has one other vital effect which is essential for sequential morphs. Internally, a particle being morphed has an item of data which shows how far it has progressed towards a full morph. This tells the Generator how far to move the vertices of the object to morph towards the target; when it reaches 100%, the object will be identical to the morph target object. Now imagine what would happen if we changed the morph target to a different object. The Generator would see that the particle has been fully morphed and would generate the second target instantly; the effect would be to snap the generated object directly to the second target without any kind of transition. By turning on this switch the morph progress is reset to zero, which is essential if we want to carry out another morph.

So, we now have the first morph working and when it is complete it will switch the Generator so it generates the first morph target directly. At this point we can turn off the modifier - in fact, we must turn it off to stop it from trying to morph the object to the first morph target. We can do this with an Action object, because the modifier is in action-controlled mode. We don't have to use additional Question objects for this because the Morph modifier itself can call actions directly. Go to the **'Actions'** quicktab in the modifier and click the **'Add Action'** button. A new Action object is added to the scene. The settings require some changes which are:

- change **'Action Type'** to **'Direct Actions'** - this gives access to the action we want
- change **'Direct Actions'** to **'Control Morphing'** - so that this Action can control the morphing
- change **'Mode'** to **'Control Modifier'** - so we can actually control a modifier rather than using the Action to affect a particle directly
- drag and drop the **'Morph to Target 1'** modifier into the **'Morph Modifier'** link field
- change **'Effect on Particle'** to **'Modifier will NOT Affect Particle'** - basically, this turns off the modifier for a particle when this action is triggered, which will happen when the morph is complete

At this point we should rename this Action object because we will have more than one of them - again, a suggestion would be to rename it **'Turn off Morph to Target 1'**.

The Action interface should now look as shown in Figure 9.3.

Playing the animation at this point won't look any different. All we have done is turned on the modifier at frame 10, and when the morph is complete the modifier will force the Generator to generate the morph target directly and then turn itself off for that particle.

To carry out the second morph, we need another Morph modifier. You can copy and paste the existing one, then rename it to **'Morph to Target 2'**. The only other changes are:

- turn on the **'Change Morph Target'** switch
- change the **'Morph Target Index'** setting to '2' so that the morph target is the second object in the morph targets list
- delete the Action 'Turn off Morph to Target 1' from the **'Actions on Morph Completion'** list

Now all we need to do is turn on this new modifier, which needs a second Action. Copy and paste the first Action object, rename it to 'Turn On Morph to Target 2' and in its settings make the following changes:

- change **'Effect on Particle'** to **'Modifier Will Affect Particle'** - we want this modifier to start working
- drag and drop the **'Morph to Target 2'** modifier into the **'Morph Modifier'** link field

To trigger this action, drag and drop it into the **'Actions'** list in the first modifier, so there are now two actions in the list which should look like Figure 9.4.

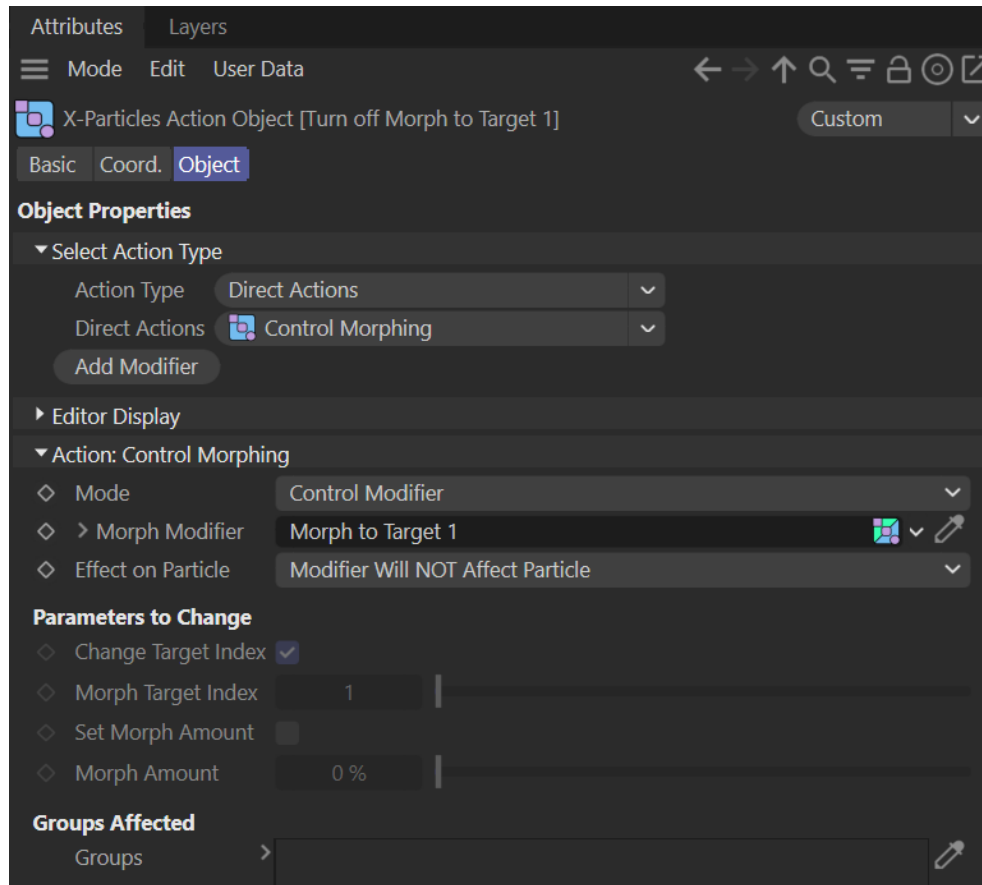


Figure 9.3. Settings for the 'Control Morphing' Action object

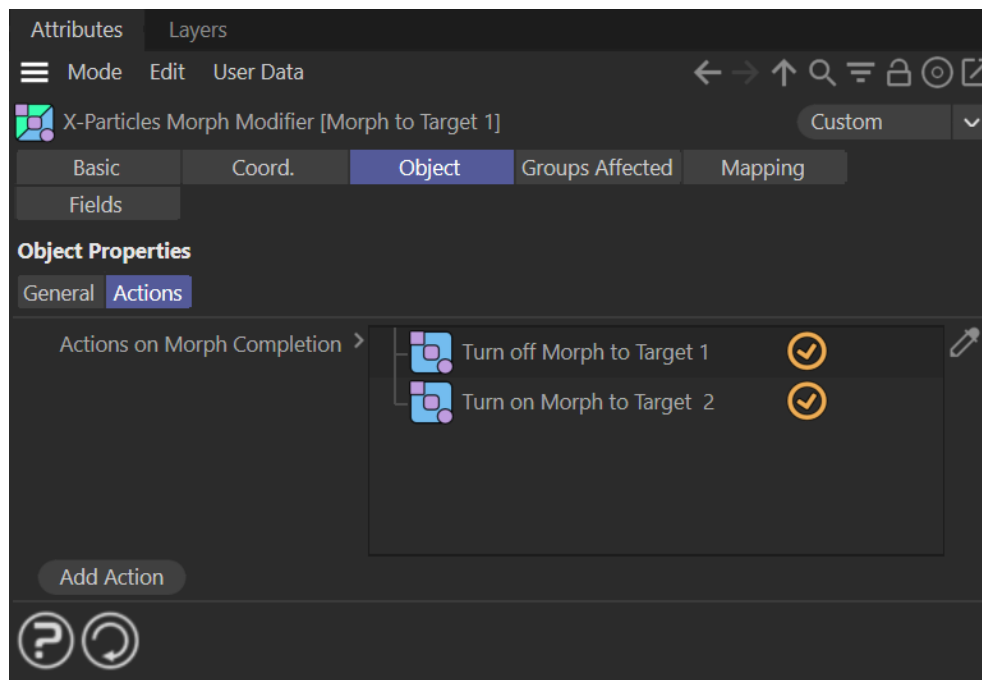


Figure 9.4. Action objects in the Morph Modifier

You can see that the sequence of events is that when the first morph is complete, the first modifier turns itself off and then turns the second modifier on. To add further stages you would add a new modifier for each morph and two actions to turn the current modifier off and the new one on.


There is no limit to how many stages you can have. Take a look at the [multiple morph animation](#) on the book's website. This is the same animation from pyramid to cube and then to reversed pyramid, but then repeats indefinitely using more Actions and a third Morph modifier. The example scene file is [cb9_generator_multimorphs.c4d](#).

I hope this discussion of morphing has been of some interest. It's quite long but I wanted to go into more detail than was possible in the XP reference manual.

9.6 Modifiers affecting the Generator

The only modifier other than the Morph modifier which acts on the Generator is the Geometry modifier. This is a very simple modifier which can change the object being generated. In the example file [cb9_generator_geommod.c4d](#), the Generator initially generates the first child object only - a Cube. When particles enter the zone of effect of the spherical falloff, the modifier changes the index of the object to generate to be the second child object, so the Cube appears to turn into a Sphere - but it doesn't morph to a sphere, it simply changes the object being generated.

The only other settings apart from the index of the child object to generate are a switch **'Random Threshold'** and a **'Threshold'** value. The **'Threshold'** value is only available if the switch is turned off, and then it works in exactly the same way as the threshold value in the Morph modifier (see section 9.5.3 'The Morph modifier' above). If the switch is turned on, the particles still have a threshold value but it is randomly generated for each particle. This is the default setting which is to maintain compatibility with older scenes.

 It's important to realise that the Morph and Geometry modifiers do NOT change the Generator itself. You won't see any changes in the Generator object's interface if you use these modifiers. When an emitter is linked to a Generator, some additional data is created for each particle, such as which child object of the Generator (from what could be several different child objects) should be generated. It is the particle data which the modifiers change, not the Generator itself.

9.7 Generator tags

The only tag directly relevant to the Generator is the Object Link Tag. This is probably one of the least-used tags in X-Particles! It was added for a specific purpose but may be useful in other scenes, so it is worth knowing how to use it.

9.7.1 What it does

The tag lets you link other objects to the object being generated. These linked objects are generated when the main object is generated but do not need to be child objects of the Generator (although they can be). You can add multiple linked objects and use a variety of methods to render lines connecting the main object to the linked one(s).

You might wonder, connecting lines aside, what the point is. You could after all create a null object, drop the main and linked objects in that, and make the null object a child of the Generator. This would not give the same result at all. The linked objects are linked to the main generated object - if that moves, they move. That lets you set up animations like this simple demonstration on the book's website. The scene file for that animation is in the download archive as [cb9_generator_linktag_anim.c4d](#).

9.7.2 Using the tag

This is very simple. Briefly, you should:

- add an Object Link Tag to the Generator
- create the objects you want to link (and if desired, make them child objects of the Generator - this is optional, but if not done the linked objects will remain visible in the scene and you will have to move them out of the way so they don't appear in the rendered image)
- drag and drop the objects to link into the **'Objects to Attach'** list in the tag interface
- very importantly, set the **'Use Multiple Child Objects'** menu in the Generator to **'First Child Only'** - any other setting will not work (note that adding this tag to the Generator will automatically set this menu option for you)
- choose how you want to render any connecting lines

That is all you need to do.

9.7.3 Line options

You don't have to have lines connecting the main object to the linked ones, but usually you would probably want to do so. You can do this in a number of ways, and these are demonstrated in the example file [ch9_generator_linktag.c4d](#). In this file, the initial setting for **'Draw Connecting Lines'** is the default **'None'**. You can try each option in turn, but note that to use the Sketch and Toon option you need to turn on the 'Sketch and Toon' render effect switch in the C4D render settings. Also, the XP material, hair, and sketch and toon options will only show when the scene is rendered.

9.7.4 Other options

These are covered in the X-Particles manual, but one worth mentioning is the **'Use Generator Scale'** switch. In the example file [ch9_generator_linktag.c4d](#), try turning that off and you will see what it does better than any explanation! If you leave it off, the default state, the linked object will be generated at its own size, but if it is on, the scale used by the Generator will change the size of the linked object.

Summary

This completes our discussion of the Generator object. To generate geometry from particles there is an alternative which works well when you only need simple objects, and that is the Sprite object, which is the subject of the next chapter.

Chapter 10: The Sprite Object

The Sprite object provides a convenient way to generate simple geometry as an alternative to the Generator object. It can also generate special objects such as lights, point clouds, and text, and has its own dedicated modifiers and a shader.

↓ The scene file(s) for this chapter can be found in the archive [examples_ch10.zip](#) on the book's website. Click here to [download the archive](#) (note: this is a large file because it contains the image sequence used in the animated texture example).

10.1 Using the Sprite object

All you need to do is add an emitter to the **'Emitter'** link field of the Sprite object and run the animation; the Sprite object will immediately add geometry to the particles. You don't need to provide an object to be generated.

10.1.1 Cinema 4D parametric objects

Many of the objects look like the Cinema 4D primitive objects, because that is exactly what they are. The Cube, Placard (actually the Plane object), Sphere, Cylinder, Capsule, Cone, Pyramid and Disc are the same primitive objects C4D produces, and in the Sprite object they sometimes have a subset of the options associated with each primitive. For example, the Cylinder sprite has an option to turn caps on and off but it doesn't have any of the caps options, and there are other options in the full C4D object which aren't necessary for this purpose.

There are some differences; all the objects are set by default to face the positive Z axis, or if they 'point' in a direction - such as the long axis of the Cylinder - this is set to point along the +Z axis. You can change this but it ensures that the sprite is aligned with the default particle direction of travel.

10.1.2 The Cross and 3-Plane objects

These are two additional parametric objects the Sprite object can generate. They don't really need an explanation - just try them out to see what they look like. Note that there is a missing feature in the 3-Plane object: unlike the C4D primitives and unlike the Cross object, the 3-Plane object is not UV-mapped, which limits its usefulness for displaying textures.

10.1.3 Light objects

Instead of parametric geometry, the Sprite object can also generate lights. These are C4D Light objects and if you use them you will see that they only carry a small subset of all the many parameters of the full Light object.

By default the lights will not illuminate the scene but will be visible, since this is the most likely reason to generate them. They can be set to illuminate the scene though, and then they behave as point (omni) lights, with shadows and falloff if desired. See the example file [ch10_sprite_lights.c4d](#) for the results that can be obtained. Render out the full 90 frames to see the changing shadows the light particles produce.

10.1.4 Points

The Sprite object can produce vertices without polygons if required, and it can do so in two ways. If you select 'Points (Individual)' the Sprite object will actually generate one spline per particle, with each spline containing one vertex. But if you choose 'Points (Point Cloud)' you will get a single polygon object with one vertex per particle but no polygons. You could, for example, add the Sprite object as a source object for the OpenVDB mesher to generate a mesh from the vertices.

10.1.5 Sprite size

How big are the objects generated by the Sprite object? There are two ways to set the size. Unlike the Generator object, the Sprite object doesn't take a child object and generate copies of that; instead it generates objects from scratch, so there is no existing object to provide a reference size and the Sprite object must set an arbitrary starting size, which is 5 scene units.

The default method to scale an object is to use the particle radius. That value will then be used as the radius of the object's bounding box, so if the Sprite object generates a cube and the particle radius value is 10 units, the actual object size will be 20 units on each axis. If you look at the generated object however, its scale will be set to 4. Why is this? Recall that the Sprite object sets the starting size of the generated object to be 5 units, so if the size of the generated cube must be set to 20 units, the scale must be

increased accordingly ($5 \times 4 = 20$).

The other method is to use the particle scale. Again, the Sprite assumes a starting size of 5 units and this is then multiplied by the particle scale; the particle radius is ignored. If the particle scale is 1 (the default) and cubes are generated, the actual size of the generated cubes will be 5 units ($5 \times 1 = 5$). The advantage of this method is that you can scale the sprite differently along its three axes.

10.1.6 Sprite rotation

There are no separate rotation controls in the Sprite object, so the sprites follow the particle rotation, if any. However, there is a button labelled **'Set Rotation to Face Camera'**. If you click this, the Sprite object will change some rotation settings in the emitter, namely:

- **'Use Rotation'** is turned on in the emitter's **'Extended Data'** tab
- **'Rotation'** is set to **'Face Camera'**

That's it. It's there for convenience and you could do the same thing manually.

10.2 Text sprites

These are more complex than the other types of sprite so need a section all of their own.

The Sprite object generates text by creating a Text spline for each particle and making the spline the child of an Extrude object. But what text does it generate? This is controlled by the **'Text Mode'** menu, which has several options:

10.2.1 Alphabet

Each sprite is a single letter of the alphabet. The **'Alphabet Mode'** setting lets you choose whether the letters are generated randomly or in sequential order (that is, the first sprite is 'A', the second 'B', and so on). The **'Case'** menu selects between all upper case, all lower case or a random mixture of both.

10.2.2 Numbers

Similar to 'Alphabet' but for numbers; again, a random or sequential output is available. The difference is that you can select the range of numbers to output, so if you want only to generate numbers between (say) 53 and 287, you can do that.

10.2.3 Alphabet and Numbers

This combines the two, with a letter followed by a number - so you get B24, X731, and so on. All the options discussed above are available.

10.2.4 User Text

Enter some text in the **'Text'** field and that text will be generated for each particle. The only other option is a switch labelled **'Sequential Output'**. If this is turned on, the Sprite object will generate one letter from the text you entered, in sequence. For example, if the text is 'X-Particles' then the first particle will show 'X', the second a hyphen, the third 'P', and so on.

10.2.5. User Text and Numbers

As for 'User Text' but with a number attached, as with 'Alphabet and Numbers'.

10.2.6 User Text (Phrases)

This mode lets you output whole phrases rather than single letters or a single phrase. In the **'Text'** field, what you do is enter phrases separated by a single character such as a semi-colon, comma, etc. (this is selectable from the **'Separator'** menu). The entered text might look like this:

Emitter;Generator;Sprite;Trail

The Sprite object will then select one of these phrases and generate the whole phrase, not just single letters. The separator character will not be generated. The phrase is either chosen randomly, or if **'Sequential Output'** is turned on the first phrase is generated first,

then the second one and so on.

10.3 Rubble sprites

'Rubble' sprites are a simple way of generating small irregular shapes which might be used, for example, as bits of rubble in a scene. There are a number of options which can be used to tweak the result.

10.3.1 Rubble shape

There are a number of these and you will immediately recognise them as variants of the C4D Platonic primitive. You can experiment with them, but in general the more polygons in the shape the more appealing the result. Icosa or Bucky give good results.

10.3.2 Subdivision

You can subdivide the generated object to increase the polygon count. The default is set at 1, which subdivides the object once. The reason for doing that is simple; to make the objects less regular, the polygons are deformed by the Sprite object when the object is generated and you will see a greater effect of this with more polygons. For example, even with two or three subdivision levels and deformation set to 250% or more, Tetra or Octa will still look like their base shape. Icosa on the other hand can start to generate realistic-looking rocks or boulders. Note that you can set the level to zero, which removes any subdivision.

Generally speaking, subdivision levels of 2 or 3 give the best results; higher than this starts to give very small polygons and a very faceted appearance, as well as slowing down the viewport.

There is also a switch '**Use Smooth Subdivision**'. All this does internally is use a feature of the Cinema 4D SDK to 'smooth' the effect, but in practice what it tends to do is convert the object into a more spherical shape.

10.3.3 Deformation

This is the amount of polygon deformation. The higher the level, the more irregular the shape. This is a matter of personal preference; I tend to use 200-250%.

10.3.4 Smoothing

A problem with these objects, especially those types with low polygon counts, is that they look faceted, even when deformed, which is undesirable. You can mitigate this in several ways.

Phong tag

There is an option to add a phong tag to the generated objects, which is the most cost-effective way in that it does not increase the polygon count. It may be less effective, however, with high levels of deformation and/or higher subdivision levels.

Use Smooth Subdivision switch

This doesn't alter the faceted appearance much, if at all, and for simply smoothing the object this option isn't very useful.

Smooth Rubble

What this does is very similar to adding the generated object to a Subdivision Surface object. Although it increases the polygon count, it is the best way to smooth the rubble sprites. You will still need a phong tag even if using this method.

10.4 Sprite textures

10.4.1 Using textures

You can apply a texture to the sprites by dropping the material onto the Sprite object to create a texture tag in the usual way. Then all sprites will show that texture. But there is another way. Instead of adding to the Sprite object, drag and drop it in to the '**Texture List**' in the Sprite object's '**Textures**' tab. This won't create a texture tag on the Sprite object, so you also have some basic parameters - the projection and tiling - which will be used for all materials in the list. Also be aware that if you have a texture tag on the Sprite object and one or more materials in the list, the ones in the list always take precedence and the tag is ignored.

The reason for this list is to allow you to put different materials on different sprites. This is controlled by the **'Choose Material to Apply'** menu, which has three entries:

- Material 1 Only - all sprites are given the first material in the list
- Random Selection - each sprite has a randomly selected material from the list
- Select By Index - you can choose which material from the list to use, starting with 1 (the first material in the list)

Whichever you choose, you can change the material to be used on a sprite with the Sprite Control modifier (see below).

10.4.2 Using a Multishader

You also have the option to use a Mograph Multishader shader. To do this, you need to work through these steps:

- create a material with a Multishader in the Color channel
- add two or more shaders to the Multishader
- drag and drop the material into the **'Texture List'** in the Sprite object
- in the Sprite object, choose how the Multishader will select which shader to use in the **'Multishader Control'** menu

A very simple example file, [cb10_sprite_multishader.c4d](#), can be found in the download archive for this chapter. Note that in C4D version R23 or later, you won't see the multishader effect in the viewport, only on rendering.

When you play the animation, each sprite will have a texture with a different shader from the list in the Multishader. The **'Multishader Control'** menu lets you choose how the Multishader will select a shader. When set to **'None'** the Multishader is not used, so it must be given some other value. If you choose **'Greyscale'** the texture is chosen randomly. Internally, the Sprite object assigns a random greyscale value to generated sprite and that value is used by the Multishader to select the texture to use. For a completely random selection it is recommended that you leave the Multishader **'Mode'** setting at the default **'Color Brightness'**.

You can get slightly different results by choosing **'Color'** instead and changing the Multishader **'Mode'** to **'Color Red'** (or blue or green). The Sprite object gives each generated sprite a random RGB colour which is then used by the Multishader to select a texture to use.

Finally, if you choose **'Use Gradient'** you can set up a gradient either in colour or greyscale, and a value is randomly chosen from the gradient and assigned to each particle. This lets you skew the selection of textures away from a random selection. If you look at the example file and change **'Multishader Control'** to **'Use Gradient'** you will see that a gradient is used which causes most of the sprites to receive the red texture with very few getting the green or blue textures.

10.4.3 Using animated textures

It is possible to use animated textures in the Sprite object and have each sprite start at a different frame in the animation. It's quite complex to set this up and the manual explains it well, but it is worth repeating it here.

The first thing you need is an animated texture. This can be a sequence of still images or a single movie file, but it is strongly recommended that an image sequence is used. This is much faster - if you use a movie it can take a very long time for Cinema to isolate each frame in the movie and send it to a new layer. In this example I've created a fairly long sequence from an animated shader and you can see the [animated shader effect](#) on the book's website. This was then saved out as a series of still images and this is included in the example file [cb10_sprite_animtexture.c4d](#).

The next process is to add this image sequence to a Multishader, so in a new material add a Multishader to the desired channels (in this case I used Color and Luminance with the same image sequence in each). You can add the entire sequence to each channel by clicking **'Add From Folder'** in the Multishader and navigating to wherever your image sequence is stored. The Multishader will add each frame as a new texture so you end up with (in this case) a list of 101 textures in both channels. I also added a single image to the Alpha channel so that the image can be applied to a Placard sprite. Again, this is included in the example file. The result in the Color and Luminance channels should look like Figure 10.1.

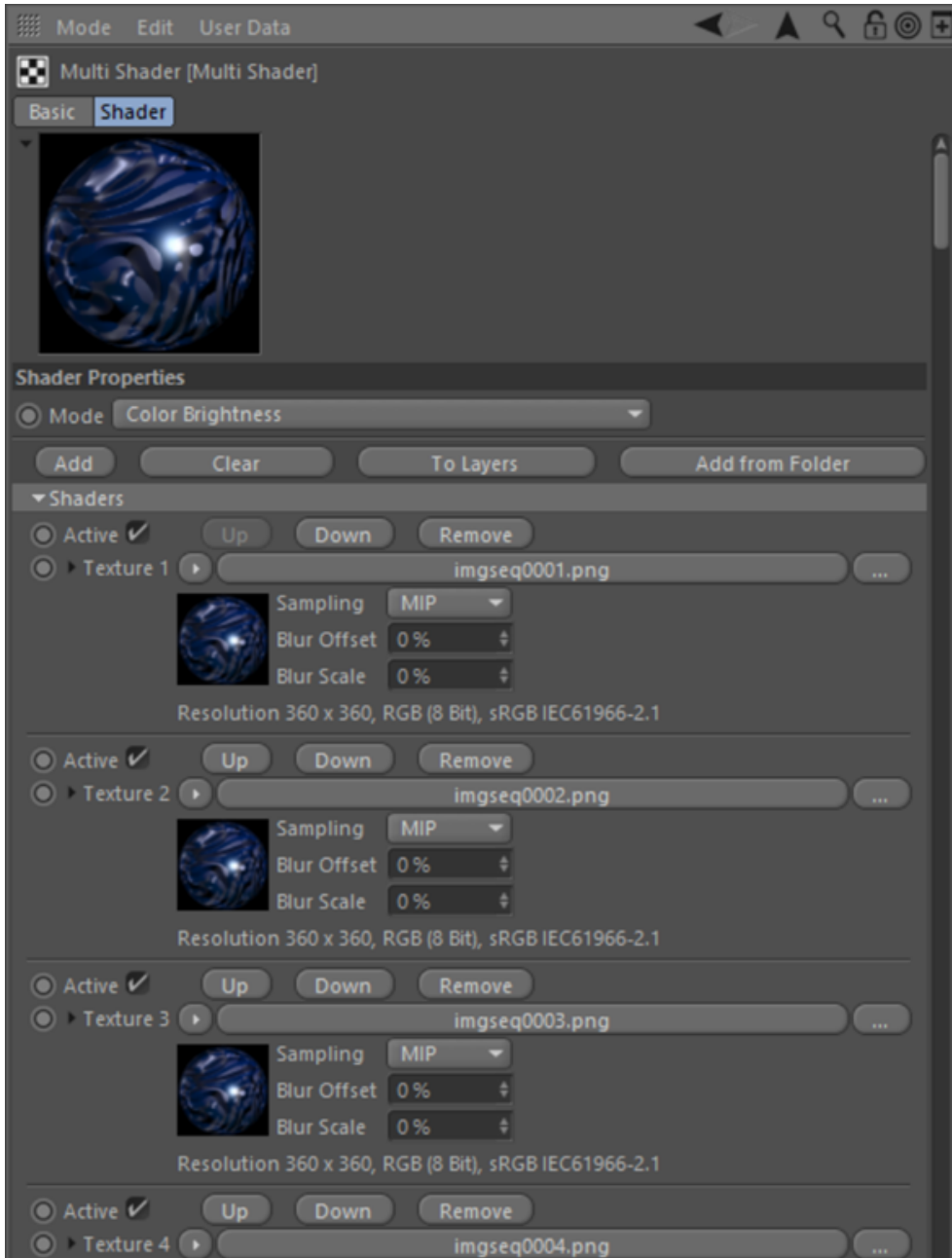


Figure 10.1. Setting up the Multishader for animated sprite textures

The material is then added to the Sprite object, and in the Sprite object **'Multishader Control'** is set to **'Greyscale'**. In the example file, if the animation is played at this point the texture is animated but the texture on all the generated sprites starts at the same frame in the image sequence, so they all look the same at any time in the animation. You can see this if you play some frames from the example file in the viewport and render that frame - you will see that all the sprites look identical. If you look at the Color channel Multishader in the attribute manager, and click on any of the textures, there is a tab labelled 'Animation' and in that you will see that all the textures show the 'Movie Start Frame' to be 1.


It would be better if they all started at different point. In the example file, turn on the switch **'Random Start Frame'**. Play a few frames then look in the Color channel Multishader again. If you click on 'Texture 1' you see that now the 'Movie Start Frame' is 14 (at least, it is on my system!) and for 'Texture 2' it is 93, and so on. This means that any sprite given 'Texture 1' from the Multishader will start the animation at frame 14 of the image sequence rather than 1; any sprite given 'Texture 2' will start its animation from frame 93, etc.

You can also set the **'Animation Mode'** in the Sprite object so that the texture continues to animate even when it reaches the final frame in the sequence. I've chosen **'Ping-Pong'** as this ensures there is no flicker when the image sequence reaches its final frame but simply reverses the animation. You can see the [final animation](#) on the book's website.

10.5 The Sprite Shader

This is a shader which lets you use particle parameters such as age to control the texture of a generated sprite. To do this, you must link the shader to an emitter, because that emitter is the source of the particles whose data you will use to control the texture.

Note that this is a channel shader, not a material like the XP Material, so you can use it any channel of a material which is then applied to the Sprite object. What this means is that in theory you could have two emitters in the scene, one to link to a Sprite object to generate the sprites and another (perhaps an invisible emitter) with a while different set of parameters which will control the sprite textures.

 Note that the effects of the shader are not visible in the viewport, only when rendered (either to the viewport or picture viewer). This is because the shader requires information that is only available from C4D at render time.

10.5.1 What can you do with the shader?

With all the default settings, the shader will assign a random colour to the particle. Or, by changing the **'Mode'** setting you can use the particle's own colour instead or select a random colour from a gradient you specify. These don't look very exciting options. After all, if you didn't use the shader the sprites would automatically take the particle colour anyway, and that can itself be set to a random colour.

But remember, this is a channel shader, so you aren't restricted to the Color channel of a material. You could add it to the Transparency channel for instance, and set the sprite's transparency from its own colour. Or you could use a second emitter with particles given a random colour and that would then drive the sprite transparency. The example file [ch10_sprite_shader1.c4d](#) shows how this could be done. If you play a frame or two, then render the result, you can see that the sprites have a random transparency setting which is driven by the second emitter's particle colours, which are set to random greyscale values. (Don't worry that the sprites seem to be invisible in the viewport. This is because the transparency channel is activated in the material attached to the Sprite object; they will render as normal.) The scene also uses a Color modifier to change the brightness of the second emitter's particles' colour. If you render this test file for the full 90 frames you can see the sprites becoming increasingly transparent with time.

You can also use modes which are dependent on certain particle parameters. Colours are selected from a gradient depending on the value of the parameter. For example, if **'Mode'** is set to **'Parameter-Dependent'** and **'Particle Parameter'** is set to **'Particle Life'** then the sprite will have a colour from the left hand end of the gradient when the particle is first created and from the right hand end when it reaches the end of its life.

The **'Age'** and **'Speed'** modes are slightly different because instead of a gradual change over time or as the particle speed changes, you simply set a cutoff point when a change in the colour will occur. For example, see the example file [ch10_sprite_shader2.c4d](#). When you render the animation you will see that the sprites are fully visible before frame 30 but abruptly disappear at that point because the parameter is set to **'Age'** and the **'Min Age'** to frame 30. At that point the colour changes from the colour at the left hand end of the gradient to the one at the right hand end - it's a single change, there is no intervening transition. You can do the same with speed in this file - change **'Particle Parameter'** to **'Speed'** and now, when the particles reach a speed of 150 units the sprites will disappear (i.e. become fully transparent).

Then you have the **'Age Range'** and **'Speed Range'** options. Here, you can set both minimum and maximum values for age or speed. In the example file if you change **'Particle Parameter'** to **'Age Range'** you see that the minimum and maximum ages are 30 frames and 60 frames respectively. The whole gradient is used in this mode so the sprites will start to fade when the particles are 30 frames old and be completely transparent at 60 frames old. You can do the same with **'Speed Range'** and then the sprites will start fading at a speed of 150 units and be completely transparent at a speed of 250 units.

The final mode is **'Modifier Sets Value'**. Here, the colour value is set by a special modifier - the Sprite Shader Modifier- rather than directly by the shader. It's more relevant to discuss this here rather than in the Sprite Modifiers section.

10.5.2 Modifier Sets Value greyscale mode

This requires a Sprite Shader Modifier. All this modifier does is to set a special colour value which is held internally by the particle. If the shader is set to **'Modifier Sets Value'** mode it will add that internal colour to the colour the shader generates itself. By default that is 100% white but you can change it by adding a shader to the **'Texture'** link. For example, if you add a Color shader set to pure black, the Sprite Shader will return black instead of white.

The combined colour is then used in exactly the same way as any other - it's only a matter of where the colour comes from.

The point of all this is that the Sprite Shader Modifier sets that internal colour value which is added to the Sprite Shader colour. Take a look at the example file [ch10_sprite_shader3.c4d](#). This uses the default settings of the modifier, which will increase a greyscale value by 1% each frame. As we have seen, the shader will, by default, always return the colour white. In the Transparency channel of a material, this will cause the sprites to be completely transparent, as you can see if play a few frames and render the scene. The modifier is returning a colour which is added to the shader colour but since this is already white, the transparency won't change - it will always be completely transparent.

There are two things we can do at this point. First, we could arrange for the sprites to start as completely opaque and then become increasingly transparent. This is easy: in the example file, add a Color shader to the **'Texture'** link in the Sprite Shader and set its colour to black. This will result in zero transparency. When the animation is played, the modifier increments the internal greyscale value each frame and the shader adds this to its own colour (black). The result is that the colour moves from black to white with time, so the sprites become increasingly transparent. You can alter the speed of change by altering the **'Greyscale Rate of Change'** setting in the modifier. Try setting it to 3% and by frame 34 the sprites will be completely transparent.

There's another setting here which is important. With the rate of change at 3%, set the **'Clamp To'** colour to 50% grey and play the animation again. Now you see that the sprites always remain semi-opaque no matter how long the scene plays. This happens because internally the modifier will not increment its internal greyscale value above 50% grey, so the transparency in the material is never greater than 50%.

What if we want to do it the other way round - start with completely transparent sprites which gradually become opaque? To do this, remove the Color shader from the Sprite Shader (or change its colour to pure white). Now the sprites will be completely transparent. To make them opaque, we need to subtract the greyscale value from the shader's colour, so in the modifier, set the **'Greyscale Rate of Change'** to -3% and set the **'Clamp To'** colour back to white. You would expect that if you play the animation now, the fully transparent sprites would become visible by frame 34, but in fact they never do. This is because of the **'Clamp To'** setting in the modifier.

Internally, when the rate of change is set to a positive value, the modifier greyscale value starts from zero and increases it each frame. But it will only increase it as far as the colour in the **'Clamp To'** setting, as we saw in the above example when setting **'Clamp To'** to 50% grey. This is fine when making the opaque sprites transparent, because we want the modifier to increase the colour brightness returned by the Sprite Shader. But in case of a negative rate of change the greyscale value starts at 100% brightness (i.e. white) and decreases it each frame - but only so far as the **'Clamp To'** setting. If this is set to white the value returned by the modifier will never go below 100% brightness and the sprites remain fully transparent. To fix this all we need to do is set the **'Clamp To'** value to black and then the animation works as expected.

In summary, to use the **'Modifier Sets Value'** mode in the Sprite Shader, you need to add a Sprite Shader Modifier to the scene then set the increment or decrement values as desired. The most important thing is to set the **'Clamp To'** value correctly; if your scene doesn't work as expected, there's a good chance that this is not set to the right value.

10.5.3 Modifier Sets Value RGB mode

As well as the **'Greyscale'** setting, which increments or decrements the overall colour brightness, you can also affect the individual colour components. In the example file [ch10_sprite_shader4.c4d](#), the material Color channel has a Sprite Shader in it. This in turn uses another Sprite Shader to generate some random colours, so initially the sprites all have a random colour. What we want to do is turn them all pure blue over time.

To do this, the modifier **'Color Mode'** is set to **'RGB'**. Now the red, green and blue rates of change settings are available. To turn the sprites blue, we need to reduce the red and green values to zero and increase the blue value to 100%. This is done by setting the red and green rates of change to -1.5% and blue to +1.5%. Finally, the **'Clamp To'** colour is set to pure blue.

You can see the [rendered animation](#) on the book's website.

10.6 Sprite modifiers

There are four modifiers which affect the sprites. As with the Generator modifiers, these modifiers do not affect or change the Sprite object itself; they change the sprite data which is carried by each particle.

10.6.1 Sprite Control modifier

You can do three things with this modifier:

Generate Sprite

This is turned on by default, but if it is turned off the sprite object will not be generated. Usually you would add a Field object (or falloff, pre-R20) and turn this switch off. When the particle enters the field of effect the sprite object disappears, but the particle is still there. Unfortunately the sprite does not reappear when the particle exits the field of effect, but you can add a second modifier with its own Field object and with this switch turned on. See the example file [cb10_sprite_spritesmod.c4d](#) to see how this works.

Change Sprite

If you turn this switch on, you can change the sprite shape to any other shape except **'Rubble'**. The change will take place when the sprite enters the field of effect of the modifier. This is also demonstrated in the example file [cb10_sprite_spritesmod.c4d](#).

Note that if you want to change the shape to **'Text'**, in order to specify the text to show then in the Sprite object itself temporarily change the shape to **'Text'**, make the required changes in the **'Text'** tab, then change the shape back to whatever you want it to start as. When the shape changes, the current settings in the **'Text'** tab of the Sprite object will be used, so you need to set that up in advance.


Change Material

If you have used the **'Texture List'** in the Sprite object to hold a list of possible textures for the sprites, you can change the texture used by turning on this switch and changing the **'Material Index'**. This is an index into the list of textures, where 1 is the first texture in the list, 2 the second and so on. Again this is shown in the example file.

10.6.2 The Light modifier

This modifier only applies to Light sprites. You can use it to change three parameters of the lights - the amount of illumination, the brightness of the light if visible, and the light inner distance setting.

Each of these parameters can be changed independently of each other and the rate of change is determined by a spline control. To change one or more parameters, you must first turn on the **'Change'** switch for the one(s) you want to use. Then use the appropriate spline gadget to control the change in the parameter. With the default settings, the parameter will take the value from the left hand end of the spline when the age is zero, and the value from the right hand end when the age is at its maximum. Instead of using age, you can set the **'Spline Timing'** menu to **'Step Count'** rather than age. Then the spline will be divided into that many steps, with each step representing one frame. The value from the left hand end of the spline will be used when the step count (which is stored internally by each particle and incremented each frame) is zero and when the step count reaches the value in the **'Steps'** setting the value from the right hand end of the spline is used. The higher the **'Steps'** value, the longer it will take to reach the right hand end of the spline.

 Unfortunately there is a problem with this modifier in that with the default settings it doesn't work correctly. The modifier is one of the older parts of XP and worked correctly in earlier versions of XP and C4D. Since the modifier has not changed for some years something may have changed in C4D itself, but whatever the reason, you will need to change some settings to get it to work.

To see the problem, try the file [cb10_sprite_light_mod.c4d](#) in the download archive for this chapter. In this, the Sprite object is set to illuminate the scene and if you play and render it, all works as expected. Now enable the xpLight modifier in the object manager and you will see that not only does the light fail to illuminate but its visible brightness is greatly reduced.

To get round this problem, you should increase the **'Spline Max.'** value for the illumination and/or visibility brightness to a much higher value - say, 10,000 - and increase the value of the knots in the spline by a factor of 50 or so (so the default splines in both cases would have their knots set to 5,000). Then the modifier works as expected. However, the **'Spline Max.'** control is not available for the **'Inner Distance'** spline meaning that this parameter cannot be effectively changed by the modifier.

10.6.3 The Text modifier

This modifier only applies to Text sprites. With it you can change the text being generated by the Sprite object. The settings are much the same as in the Sprite object itself, the exception being that the mode 'User Text (Phrases)' is not available.

For details of the various text modes, please refer to the Text sprite discussion above.

Threshold

This is the only additional setting in the modifier which is not in the Sprite object. What it does is control when the modifier

actually makes the change and is used along with falloffs (pre-C4D R20) or fields (R20 onwards).

Each falloff or field returns a value from 0 to 100% for every position within the field of effect. For each particle, if the returned value is greater than the **‘Threshold’** value, the modifier will change the text as desired. If the returned value is less than the threshold, no change will occur.

You can use this with different types of falloff or field to alter when the text change occurs.

10.6.4 The Sprite Shader modifier

The use of this modifier is discussed in the Sprite Shader section above.

Chapter 11: The Trail Object

The Trail object does exactly what it says - attaches a trail to each particle. The trail itself is just a spline. Every particle used by a Trail object has a set of trail data. Each frame, the position of the particle in the 3D world is sampled and a new point at that position is added to the trail data, and that trail data is used to construct the trails spline(s). When the particle dies the trail automatically dies with it; there is no way to preserve the trails of dead particles since they don't exist any longer, and the trail data for the spline is stored in the particle.

If you create some trails and then do 'current state to object' on the Trail object, you will see that it doesn't generate multiple splines but a single spline with multiple segments, each segment being one particle's trail.

The Trail object can also create trails from moving objects rather than particles in which case you don't need an emitter at all.

You should be aware that the Trail object creates what I like to think of as 'true' spline objects. The Cinema 4D SDK can generate splines in two ways, using different functions in the SDK. One way generates a spline that can do anything a spline in Cinema 4D can do; perhaps the most obvious is that it can be rendered with the Hair material or with Sketch & Toon. The other way is easier to use but doesn't generate a true spline object, and as such, cannot be rendered with Hair or S&T; there are other limitations as well. Fortunately the Trail object generates the real thing.

↓ The scene file(s) for this chapter can be found in the archive [examples_ch11.zip](#) on the book's website. Click here to [download the archive](#).

11.1 Generating trails

This couldn't be simpler. Add a Trail object to the scene and then drag and drop an emitter into the **'Emitter'** link field in the Trail object. Play the scene and you have trails.

You will see from this that a Trail object can only use one emitter. However, a single emitter can be used by multiple Trail objects. The problem is that the trails produced by one Trail object would be identical to the trails produced by all the other Trail objects since they would all use the same trail data. There is a way around this and that is to have the emitter use multiple particle groups. Since the Trail object has a **'Groups Affected'** list, you could have each Trail object use a different group and, if desired, set the parameters in the various Trail objects differently.

11.1.1 Control switches

There are three switches which control trail generation. These are:

Trails From Particle Birth

This is on by default, and if you turn it off it trails are not generated when a particle is created. It is possible to turn on the trails on a per-particle basis by using a 'Change Trails Action'. There is an example file [ch11_trails_generate_by_action.c4d](#) in the download archive showing how this is done. In that file, a Question is used to test the particle's age, and when the age is 30 frames a Change Trails Action turns them on (for that particle - other particles less than 30 frames old are unaffected).

Trails on Streamed Particles

This should really be labelled **'Trails on Tendril Particles'**. The Tendril modifier works by generating a lot of extra particles and you normally wouldn't want these additional particles to have their own trail. However, if you turn on this switch you can do so.

In order to generate trails from tendril particles, you need to change a few settings. If you set **'Algorithm'** to **'Tendrils'** this switch is automatically disabled, so you need to select a different algorithm. This means that you no longer get a single spline connecting all the particles in the tendril, which is usually what you want when using tendrils; instead, each particle has its own trail. The result you get will depend on the connection algorithm, so it's worth trying them all to see what happens. Try the example file [ch11_trails_streamed_particles.c4d](#). The algorithm here is set to **'No Connections'** and the result is quite interesting in its own right, but try all the others and see what you get. **'Segmented Sequence'** is interesting, so is **'Nearest By Index'**.

Draw Splines

Usually you would leave this turned on, because if it is turned off the Trail object won't create any splines. The trail data is still in

the particle though, and can be tested, for example, in a Question object. If you load the example file [cb11_trails_draw_splines.c4d](#) you can see that the trails are set to a maximum length of 150 scene units. This is tested in a Question object, and when the maximum trail length is reached the particles turn red. Now turn off the **'Draw Splines'** switch and the trails splines are no longer generated, but the particles still turn red at the correct time, showing that the trail data is still present. There is no spline though: the trails won't render and you cannot use them in a Sweep object, for example.

11.1.2 Connection algorithm

This menu controls how the splines are generated from the trail data. Remember, the trail data is just a set of points in 3D space. It's how these points are connected which determines what the trail spline looks like and it is possible to connect different particles rather than using the same particle for each spline segment.

No Connections

This is the default setting and is something of a misnomer. To generate a spline the points from the trail data clearly must be connected. What this setting means is that for each particle only that particle's trail data is used to build a spline segment. This will cause a spline to be generated with a new point generated each frame and each point on the spline being the position of the particle in the frame it is generated. Each particle will therefore have its own 'tail', or trail. The final object which is generated is actually a single spline with one segment for each particle's trail.

This algorithm has quite a lot of settings you can change. The **'Length Mode'** controls how the maximum length of the trail is determined. The default is **'Time'**, so the length is determined by how much time elapses after the trail generation starts. The switch **'Full Scene Trail'** is turned on by default, so the trail is generated for the entire length of the scene (or for the particle life if that is shorter). If you turn that switch off you can decide how long, in terms of time, you want the trail to be, and you can add variation to that to give different lengths to different particles.

You also have the option to alter the **'Frame Sampling'**. This is how often the particle position is sampled to add a new point to the trail spline. The default is 1, which means a sample is taken each frame. If you increase it you will have a spline with fewer points and it may result in a smoother spline. Note that increasing the **'Subframe Steps'** in the X-Particles project settings does not result in more sampling of the particle position; the most frequently this can be sampled is once per frame.

The other length mode is **'Length'** and with this you can control the maximum length of the spline in scene units. Again, you can add variation for different lengths in different particles. Do remember that this is the actual length of the spline in scene units. For example, with the default speed of 150 units per second a particle will travel 300 units in 60 frames (at 30 frames per second) and you can see in the viewport orthographic views that the trail is indeed 300 units long. If you add a Turbulence modifier to the scene, when you play it the trail may look shorter than that, because the particle track is wandering around in three axes - but the actual spline is still 300 units long.

'No Emission Point' is an interesting setting. When a particle is created the emitter will create it at a certain point then move it forward. (The distance moved depends on whether **'Subframe Emit'** is turned on in the emitter Emission tab. If it is, the distance will be very small, though not zero; if it is off, the distance will be equal to that moved by the particle in one frame, which at the default speed of 150 units will be 5 units at 30 frames per second.) Only after the particle has moved will the first sample point for the trail spline be recorded. This doesn't matter very much in some cases but in others, such as when emitting from an object, there may be a gap between the object and the start of the trail, which is unsightly. For that reason, the Trail object can add an extra point at the actual location where the particle is created, which is termed the 'emission point'. If you don't want the emission point to be added, turn on this switch. It is in fact on by default, for backwards compatibility.

In Figure 11.1 particles are being emitted from a Plane object. The **'No Emission Point'** switch is on for the green trail so no emission point is added and there is a gap between the Plane object and the start of the trail. For the orange trail the switch is turned off so there is an extra emission point and no gap.

Finally, there are some options to freeze the particle or the trail when it reaches its maximum length. (This will have no effect if **'Length Mode'** is set to **'Time'** and **'Full Scene Trail'** is turned on.) If **'Freeze When Complete'** is set to **'Freeze Particle'** then the particle will be frozen when the trail reaches its maximum length. By default this will freeze the particle movement, spin and scale but you can turn any of these switches off if required, so you could freeze particle movement but keep it spinning, for example.

If the menu is set to **'Freeze Trail'** the particle will carry on moving but the trail won't follow it - it will just remain where it was when it reached its full length.

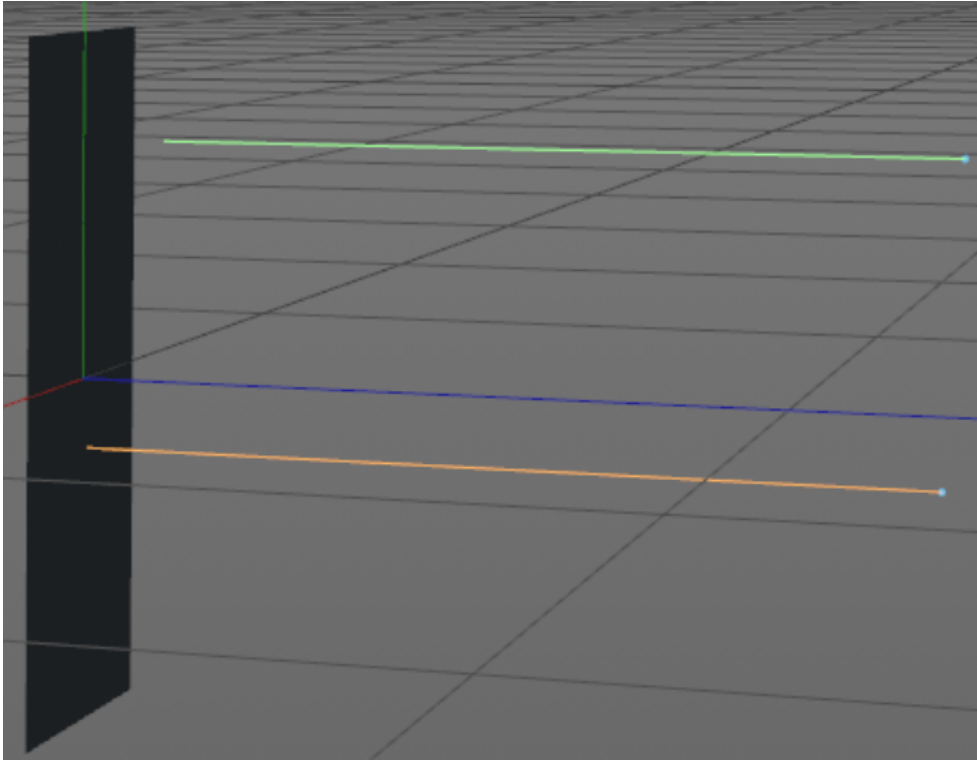


Figure 11.1. Effect of 'No Emission Point' switch

Straight Sequence

Here, a single spline with one segment is generated which connects each particle to the next in the particle sequence. Instead of each particle having its own trail, the spline is constructed with a new point added when a new particle is generated, and each point's position is updated each frame as the particles move.

The particle sequence has two options, seen in the menu **'Connect Using'**. If you choose **'Particle Index'** (the default setting) the particle with index 0 in the particle array will connect to the particle with index 1, the particle with index 1 to that with index 2, and so on. The alternative is **'Particle Unique ID'** where the particles connect in order of their ID values.

In fact, these will give the same result until you start deleting particles during the animation. The reason is that if you delete particles, the index values of some or many other particles will change (see the emitter section 'Index and ID Values' for why this happens). If the sequence is connected using the particle index, and deleting a particle causes significant changes to the index values of other particles, the trail may change radically. But if the ID values are used, which don't change during the animation, deleting a particle will certainly change the trail but probably not in a major way.

There are two other options. By default all the particles are used in building the trail but you can opt to omit some of them by setting **'Skip Particles'** to a value greater than zero. For example, if this is set to 1, the first particle will connect to the third particle, omitting the second one, then the third particle will connect to the fifth particle omitting the fourth, and so on. This can be useful if you have a lot of particles in order to smooth out the spline and make it look less 'busy'. In Figure 11.2 the emitter birth rate is 60 particles per second. On the left, the skip particles value is zero; on the right it is 4.

Finally, you can set a maximum length of the trail in terms of the number of frames the trail is generated for. To do this, turn on the **'Set Max Length'** switch and if you then set **'Length'** to 30 (for example) the trail will only be generated for 30 frames regardless of how long the scene is.

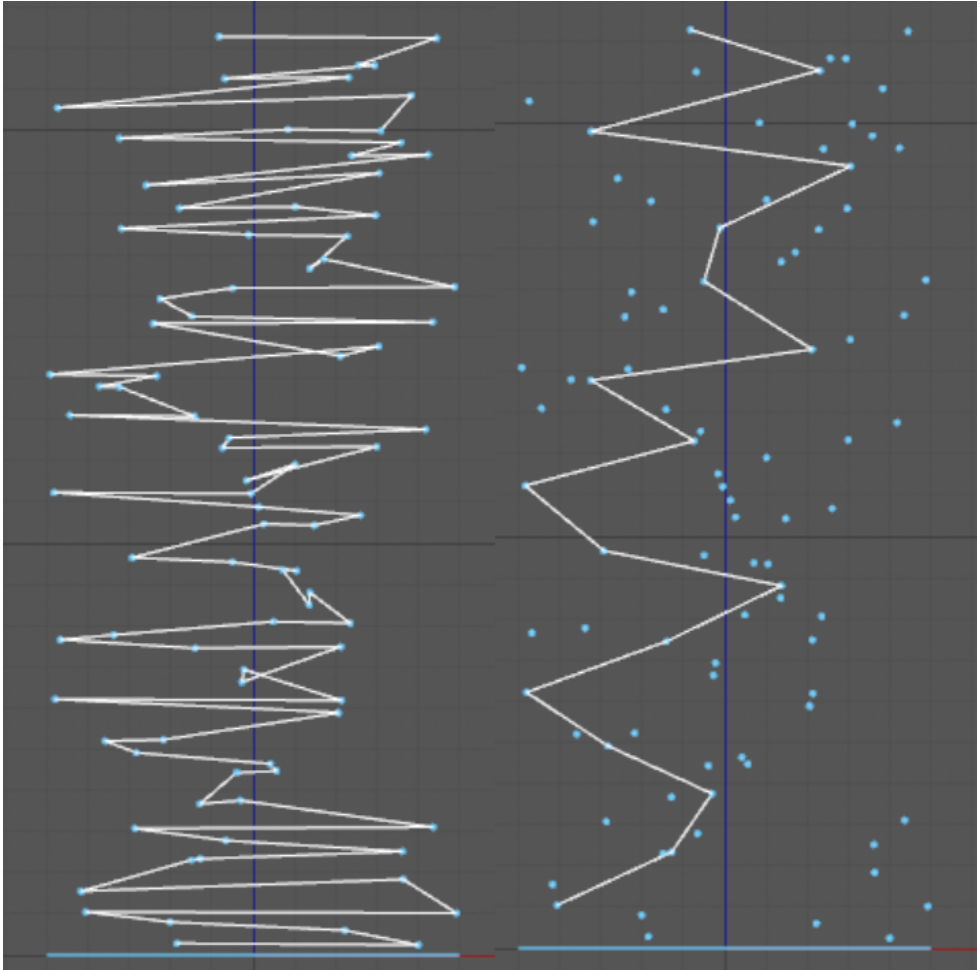


Figure 11.2. Use of 'Skip Particles' setting, comparing zero (left) and four (right)

Segmented Sequence

This is the same as **'Straight Sequence'** but instead of one single trail the spline is generated as a series of short segments with gaps between each segment. You can control the segment length and the length of the gap. The default settings mean that particle 1 is connected to particle 2, then particle 3 to particle 4, but particle 2 is not connected to particle 3 - leaving a gap. Setting **'Segment Length'** to 2 means that particles 1, 2 and 3 are connected, as are particles 4, 5 and 6, but particle 3 is not connected to particle 4.

A segment length of 1 with a gap length of 2 will cause particles 1 and 2 to be connected and particles 4 and 5 to be connected, but there is no connection from particle 2 to 3 or particle 3 to 4.

Multiple Sequences

This is also very similar to **'Straight Sequence'** except that instead of generating one long trail, you can generate multiple connected trails. The default setting using **'Alternating'** mode and a **'Sequences'** setting of 2 will result in two trails, one built using particles 1, 3, 5, etc. and the other using particles 2, 4, 6 and so on. You can increase the number of trails by increasing the **'Sequences'** value. Note that if you set this to 1, the result is identical to the **'Straight Sequences'** algorithm.

The other mode is **'Sequential'**. Now the particles are connected just as with **'Straight Sequence'** until the length reaches the number of points in the **'Length'** setting. Then there is a gap and another trail segment is generated. So for example, if **'Length'** is set to 10, particles 1-10 are connected, but then there is no connection between particles 10 and 11. A new segment is then generated from particles 11 to 20, and so on.

All Points to all Points

This is very simple: each particle is connected to every other particle, producing a very dense net of spline segments. Needless to say, if you have a lot of particles you will get a very complex spline very quickly.

Nearest by Index

With this algorithm each particle connects to a particle or particles with the closest index in the particle array to itself. The default settings will produce an identical result to the **'Straight Sequence'** algorithm using **'Particle Index'** as the connection mode. However, if you set **'Max Connections'** to a higher value, each particle will connect to more particles. For example, set to 1 (the default) particle 1 connects to particle 2, particle 2 to particle 3 and so on. But if **'Max Connections'** is set to 2, particle 1 will connect to the two nearest particles in terms of their index values - so it will connect to particles 2 and 3. Every other particle will do the same.

You can also choose to skip some particles, which works in the same way as the **'Straight Sequence'** algorithm. Finally, you can restrict the trail to certain particle groups with the **'Destination Group'** menu. The simplest way to understand this is to use the file [*ch11_trails_nearest_by_index.c4d*](#) in the download archive. Play this for 30 or so frames then switch between the entries in this menu. Note that if you choose **'Specific Group'** only the particles in the specified group will make connections, but they may do so to particles in other groups as well as their own. **'All Except Specific Group'** will only make connections between particles which are not in the specified group.

Nearest by Distance

This is similar to **'Nearest by Index'** but instead of using the particle's index in the particle array, it uses the distance in the 3D world between particles. The particle groups options are the same but there are some different options for this algorithm.

⚠ Unlike **'Nearest by Index'** the trail generated by this mode may change quite radically during the animation. With **'Nearest by Index'** the index values of the particles don't change unless particles are deleted during the animation. But with **'Nearest by Distance'** the closest particle to another particle may change depending on new particles being created, modifier effects, collisions and so on.

The first option is **'Distance Mode'**. Choosing **'Nearest Only'** from this menu will connect a particle to its nearest neighbour. Since each particle can only have one neighbouring particle which is the closest to it, each particle can only initiate one connection. However, if you look at the example file [*ch11_trails_nearest_by_distance.c4d*](#), play 30 frames or so then look at the particle with the ID value of 3, it appears to have three connections. Why is this? In this example, particle 1's nearest neighbour is particle 3, so particle 1 connects to particle 3. Particle 2's nearest neighbour is also particle 3, so it too connects to particle 3. But particle 3's nearest neighbour is particle 4, so it connects to that particle - giving the appearance of three 'nearest neighbours' for particle 3.

The second mode is **'All Within Distance'**. Here, all particles will make connections to all other particles whose distance away lies between the **'Max Distance'** and **'Min Distance'** values. You can try altering these in the example file and see what effect you get.

Finally, there is **'Max Number Within Distance'**. This is the same as **'All Within Distance'** but allows you to set a maximum number of connections each particle can make, which you do in the **'Max Number'** setting.

Cluster

This algorithm connects particles into clusters as seen in the left-hand image in Figure 11.3.

The most important setting is **'Cluster Distance'**. All particles forming a cluster must be within that distance of each other. The smaller this value, the smaller the clusters will be because fewer particles will meet that criterion. The above image uses the default distance of 50; if that is reduced to 35, this result is shown in the right-hand image of Figure 11.3.

The other setting, **'Min Particles in Cluster'** is meant to set the minimum number of particles required to form a cluster. Unfortunately in the latest version of X-Particles this feature appears to be broken.

Tendrils

This is a special mode intended for use with tendrils created with the Tendril modifier. It requires a Tendril modifier in the scene and will be discussed in more detail in the section dealing with that modifier.

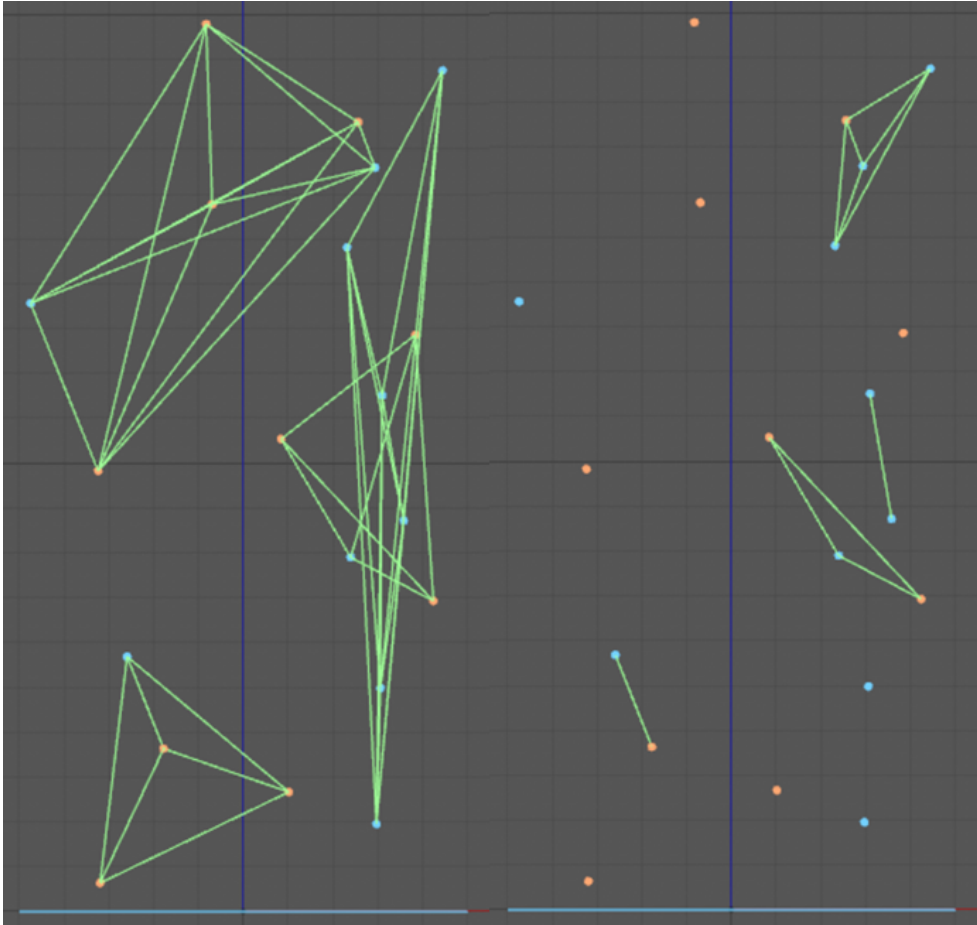


Figure 11.3. Cluster algorithm comparing 'Cluster Distance' values of 50 and 35

Constraints

This algorithm allows you to generate trails from constraints, collisions between a particle and an object, and between colliding particles.

Using particle constraints requires a Constraints object to be added to the scene. You can't normally see the constraints unless you turn them on in the emitter's Display tab. For example, enabling birth constraints would show them as yellow lines. These are only visible in the viewport and do not render. However, you can generate a trail spline from them using this algorithm and turning on the **'Birth'** switch in the **'Constraints'** section. In Figure 11.4 the basic constraints with no splines are on the left (yellow lines) and the trails are on the right (green lines). As you can see, the trail exactly matches the constraints but with the difference that you can render the trail (see below for various ways to do this).

For collisions between particles and an object, you don't need a Constraints object. Simply add a Collider tag to the object and in that tag turn on **'Connect on Collision'**. Then a constraint will be added for each particle colliding with the object. To generate trails for the constraints, turn on 'Collisions (With Object)' in the Trail object. An example file is provided as [cb11_trails_constraints1.c4d](#) in the download archive.

Finally, for collisions between particles you will need a Particle-Particle Collision object, and turn on **'Connect on Collision'** in that object. Then turn on **'Particle-Particle Collisions'** in the Trail object. You can use more than one emitter but then each one will need its own Trail object. See the example file [cb11_trails_constraints2.c4d](#) in the download archive.

Infectio

This is another special mode for use specifically with the Infectio modifier. It will be discussed in detail in the section dealing with that modifier.

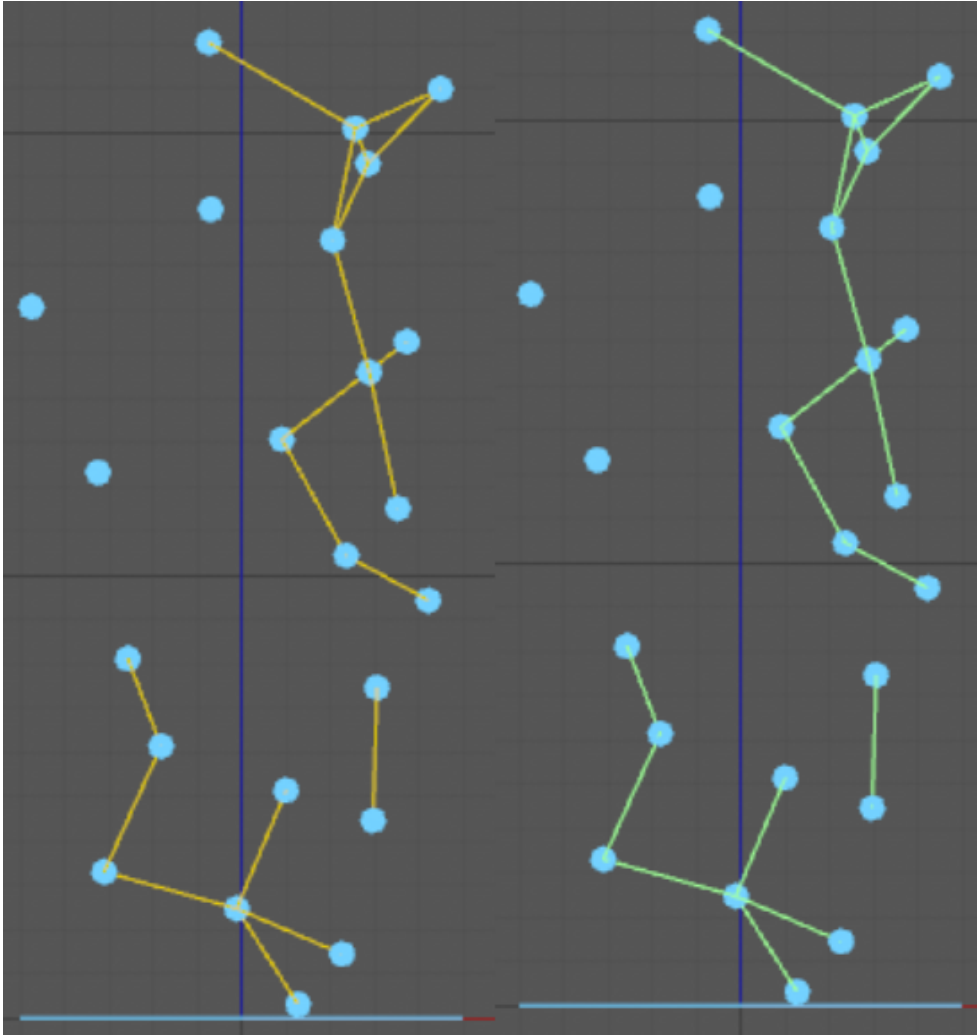


Figure 11.4. Creating trails from constraints

11.1.3 Object Trail mode

In this mode the Trail object does not require an emitter, as it generates trails from an object rather than from particles.

Object

The first thing you need is an object. This can be a polygon object or spline, or a Mograph cloner, or a Null object with child objects.

Operation

This controls how the trails are generated. If you choose **'Vertices'** then a trail is generated from each vertex in the object. In the example file [cb11_trails_object_mode_1.c4d](#), you can see a trail follows each vertex in the Cube object.

Try switching to **'Polygon Centre'** instead; now the trails originate from the centre of each polygon. For this to work, the object must actually have polygons. If you use a spline with this setting, no trails are generated.

The last option is **'Object Axis'** and now a single trail will be generated from the object's axis. This is usually, but not always, located at the centre of the object.

⚠ It is possible to use an X-Particles Sprite or Generator object as the trail source, but this will not work in **'Object Axis'** mode. This is because the object axis is that of the Sprite or Generator object, not the generated objects, and those objects don't normally move. You can even use another Trail object as the source, but then you can only use **'Vertices'** mode.

Cloners and Nulls

If you want to use a cloner or null as the source you can use all three modes, and if you choose **'Object Axis'** mode the **'Cloner and Nulls'** setting becomes available. By default this is set to **'Use Child Objects'** and this is demonstrated in the file [cb11_trails_object_mode_2.c4d](#). If you play this you can see that a trail is generated from the axis of each of the three cubes. If you change this setting to **'Use As Single Object'** then only one trail is generated, from the position of the cloner itself.

Full Scene Trail

With this switch turned on the trails are generated for the full length of the scene. If it is turned off, you can set the maximum length of the trails, either by time or actual length. This is the same as for particle trails with the **'No Connections'** setting.

11.2 Trail thickness and colour

In Cinema 4D splines don't have any thickness and the colour cannot vary at different points along the spline. X-Particles trails do have this capability, which you set up on the **'Thickness & Color'** quicktab. Colour is much easier to understand and visualise, so we'll look at that first.

Any colour changes won't be visible in the editor; to see them you will have to render the spline. The easiest way to do this is to create an XP Material and apply it to the Trail object. This will render the splines without needing any geometry.

11.2.1 Trail Color Mode

The default setting for this menu is **'Particle Color'**. What this means is that, when rendered, the entire trail is rendered in the current particle colour. If that changes during the animation the trail colour will change as well.

If the menu is set to 'Per-Vertex' each point in the trail will be coloured with whatever the particle colour is when the point is added to the spline. If the particle colour changes during the animation you will see a multicoloured spline when rendered.

See the very simple example file [cb11_trails_colour1.c4d](#). This uses a single particle whose colour is changed by a Color modifier from blue to magenta over the length of the scene. Play about 10 frames and render the scene: the trail spline is uniformly blue. Now play to the end and render; now the trail is all magenta.

Now change **'Trail Color Mode'** to **'Per-Vertex'**. Play the scene to the end and render that frame. What you have now is a spline which shows a colour gradient along its length from blue to magenta.

That's really all there is to trail colours. The only other thing you need is to be able to render the trail without geometry and with a renderer that knows how to use the trail colour data. The C4D standard renderer can do this using the XP Material, or you can use Cycles 4D if you have that. Other renderers may also be able to render the trail splines using the trail colour data.

i If using Cycles 4D, there is a preset material supplied with XP which can use the trail colours. You find this in the material manager->Cycles 4D->X-Particles menu as 'xpTrail Color Emission'. See the example file [cb11_trails_colour_thickness_cycles.c4d](#) to show this working (requires Cycles 4D).

11.2.2 Trail thickness

The trail's thickness is defined as the radius of the spline when it is rendered and is controlled by the **'Thickness Mode'** menu. By default this is set to **'Do Not Set Thickness'** so if you want to control the thickness of the rendered trail, you will need to set one of the other options. Otherwise, the thickness will be set by the XP Material if you are using that. As with the trail colour, the scene must be rendered to see the effect.

The example file [cb11_trails_thick1.c4d](#) in the download archive is set up so that you can try all the thickness modes in turn to see how they work. You'll need to enable the Scale modifier for the two 'use radius' modes.

Set From Value

This option is very simple: the thickness is set using the **'Thickness Value'** setting in combination with the **'Variation'** value if that is non-zero. The trail will always be of a uniform thickness, but if you keyframe the thickness value the trail will change in thickness as the animation plays.

Use Spline

This option will set the trail thickness from the **‘Thickness Spline’**. The start of the trail will have the thickness given by the left hand end of the thickness spline control while the end of the trail will have the thickness set from the right hand end of the spline control. This lets you vary the thickness along the length of the spline. By default the maximum thickness the spline will set is 5 units, but you can increase this with the **‘Spline Max’** setting.

Use Shader

With this option you can control the thickness of the spline with a shader; simply add the shader to the **‘Shader’** link field. Not all shaders give good results but the Noise shader works very well. The thickness value at any point along the trail is calculated by multiplying the **‘Thickness Value’** setting by the brightness of the colour returned by the shader. Since the brightness can only range from 0 to 1, the actual thickness will be anywhere between zero and the value from the **‘Thickness Value’** setting. As with **‘Use Spline’** this mode gives variation in thickness along the length of the trail.

Use Radius (Current)

This option uses the particle radius to set the trail thickness. If the radius changes during the animation, the thickness will change too. However, the thickness will always be uniform along the trail - it won't vary along the length of the trail.

Use Radius (Variable)

This also uses the particle radius to set the thickness, but the difference is that the radius is sampled every time a point is added to the spline and the thickness can therefore vary along its length if the radius changes.

The example file [cb11_trails_thick1.c4d](#) includes a Scale modifier to change the particle radius. If you enable this modifier in the object manager, you can see the difference these two modes make.

i Once the thickness has been set, to see the result you will, as with the trail colours, need a renderer which can use the Trail object thickness data. The XP Material will use the data if available.

Cycles 4D can also do this. Just make sure that **‘Thickness Mode’** is set to something other than **‘Do Not Set Thickness’** and the thickness data will automatically be used by Cycles 4D. You will need to apply a material to the spline to actually see it, but if you aren't concerned about the colour this can be a simple object material instead of the 'xpTrail Color Emission' material referred to above. Note that, if you also add a cyCurve tag to the Trail object, this will override the thickness data and set the thickness itself. To prevent this, turn off **‘Set Thickness’** in the cyCurve tag.

The example file [cb11_trails_colour_thickness_cycles.c4d](#) shows the thickness data used in Cycles 4D as well as the colour (requires Cycles 4D).

11.3 Other options

11.3.1 Spline quicktab

This tab lets you change the type of spline generated by the Trail object. By default a linear spline is produced but you can change this to any of the types supported by Cinema 4D.

The other options are the same as those found for any non-primitive spline (primitive splines have their own settings which may or may not include the settings for spline type, intermediate points, etc.).

11.3.2 Groups Affected tab

In this tab you can control which particle groups are used by the Trail object. If you drag a Group object into this list only the particles in that group will be used to for the trail.

There is a file in the download archive, [cb11_trails_groups.c4d](#), demonstrating the use of groups. This scene has one emitter which emits particles into three groups. Each group is identical in colour, using 'Gradient (Parameter)' mode with the default blue to white gradient. They only differ in the particle shape. If you play this scene, you can see from the particle shapes that the particles are being emitted randomly into one of the three groups.

There are three Trail objects in the scene, each using the same emitter but only using one of the particle groups. The Trail objects have different colour and thickness data settings (see above for more discussion about this) so when you render the scene the result depends on which Trail object a particle uses, and that in turn is determined by the particle's group. This is the sort of result you will see:

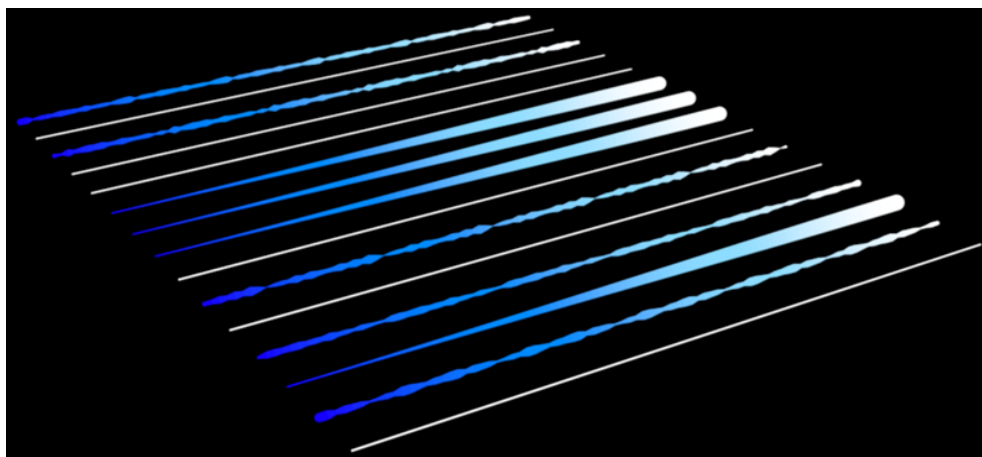


Figure 11.5. Using particle groups with Trail objects

Enabling/disabling groups in the list

In the **'Groups'** list, if you click the yellow tick icon for a group to change it to a blue dash, that group won't be added to the internal list of groups which the Trail object will use. This has certain implications.

Consider what happens if you have three groups in the scene and add one of them to this list. From then on, only the particles in that group are used in the trails. However, if the group's icon in the list is changed to a blue dash, then when the Trail object decides which group(s) to use, that group is not included in its internal list. That is what you would expect. The problem is that there are then no groups in the (internal) list and what happens then is that the Trail object refers to the setting in the menu **'When No Groups In List'**. The default is to affect all particles, so the Trail object now uses the particles in all groups, when what you probably expected was that it would use none at all. You can change that menu to **'No Particles Affected'** to get the result you expected.

I emphasise this because if what you want to do is exclude a specific group from the trails you can't just add it to the list and change its icon to a blue dash. What you need to do is add all the groups to the list, then change the one you want excluded to have a blue dash icon. This is not very user-friendly but it's the way it works.

11.4 Rendering the trails

The trails aren't very useful unless you can render them in some way. There are several ways to do this but they can be divided into two groups: those that generate some geometry using the trail as a source and those that render the spline directly without additional geometry.

11.4.1 Generating geometry

The Sweep object

You can use a Cinema 4D Sweep object and sweep another spline shape along the trails. Unfortunately the Sweep object knows nothing about the Trail object colour or thickness data so won't colour or scale the trail splines accordingly. However, you aren't restricted to circular trails as you can sweep other shapes along the trail.

The xpSplineMesher object

This is part of X-Particles and is an advanced version of the Sweep object. Most of its capabilities aren't needed for simply generating some geometry using a Trail object but it does have one significant advantage over the Sweep object: it can use the Trail colour data.

To use the object, add it to the scene then drag and drop the Trail object into its **'Objects'** list. You can leave most of the other settings alone but you might want to change the **'Size'** value and increase the **'Subdivisions'** value to 1 (otherwise you get a rather blocky, square result).

To use the trail colour data, make sure that the switch **‘Vertex Colors’** is checked, which it is by default. This will add a C4D vertex colour tag to the spline mesher object. You can then use this tag in a Vertex shader. The example file [ch11_trails_splinemesh.c4d](#) shows how this is done. The spline mesher cannot, however, use the Trail thickness data.

11.4.2 Rendering splines without geometry

To do this you need two things:

- a material with which to render the spline
- a render engine which supports the Trail colour and/or thickness data

The XP Material

This is a material supplied with X-Particles. Originally intended to render particles without geometry, it can also render splines (any spline, not just the Trail object splines). To use it, simply add the material to the Trail object and render; the material works with C4D’s standard renderer.

By default the material will use the particle colour to render the trail splines, which is exactly what you need. In the Trail object, if you change **‘Trail Color Mode’** to ‘Per-Vertex’ then the particle colour at each trail point will be used by the material. This renders changes in colour along the trail, if the particle colour changes during the animation.

The XP Material will also use the trail thickness data, if present. You only need to change the **‘Thickness Mode’** to something other than **‘Do Not Set Thickness’** and the material will use this data automatically.

As an example, here are rendered trails with a colour gradient from blue to red and with thickness controlled by a Noise shader:

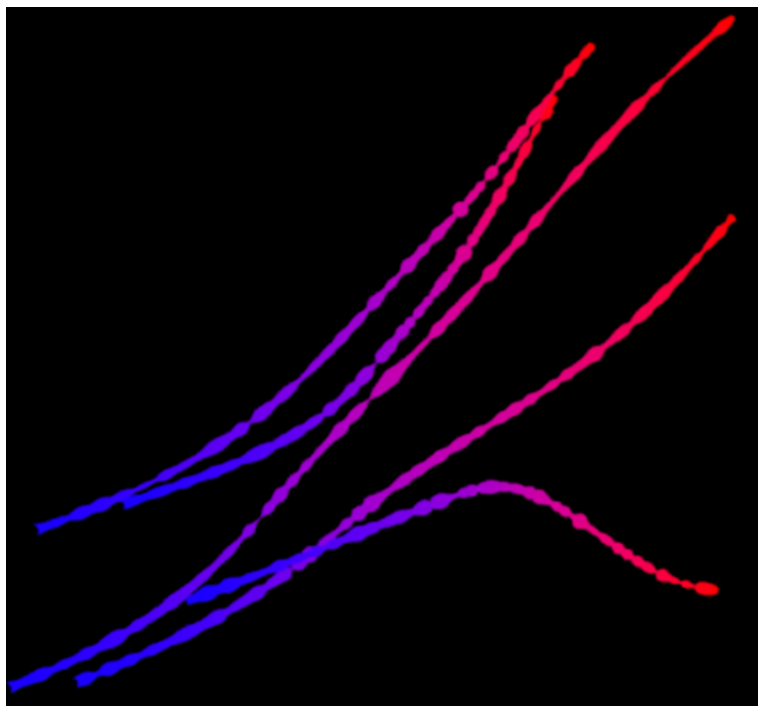


Figure 11.6. Trails rendered using the XP Material

Cycles 4D

You can also use Cycles 4D, if you have that (it’s part of the Insydium Fused collection). There is an example file in the download archive, [ch11_trails_colour_thickness_cycles.c4d](#), showing how this works.

Hair and Sketch & Toon


If you have these modules of Cinema 4D, you can use either of them to render XP trails. However, they don’t know anything about the XP trail data so you are restricted in terms of colour and thickness to whatever these shader systems can do. Please refer to the C4D documentation for details.

11.5 Other objects affecting the trails

There are two objects which can affect the trails: the Trails Modifier and the Trails Deformer.

11.5.1 The Trails Modifier

The only modifier affecting the Trail object is the Trails Modifier. This has only one purpose: to reduce a trail's length (Shrink mode) or make it longer (Expand or Grow modes).

 This modifier only works on Trail objects where the algorithm is set to **'No Connections'** - that is, where each particle has its own individual trail. No other connection algorithms are affected.

Shrinking trails

This is the default mode of operation (**'Operation'** is set to **'Shrink'**). When the particle enters the modifier's field of effect, it will shrink. The speed with which it shrinks is controlled by the **'Rate of Change'** parameter so that the higher this value the faster it shrinks. A rate of change value of 1 will shrink the trail by one point in the trail each frame. Note that **'Rate of Change'** is not a time or physical length setting, it is the number of points by which the trail length is reduced each frame.

You can also shrink the trail more slowly than one point per frame with the **'Interval'** setting. This is the time which must elapse between each shrink action. For example, if **'Rate of Change'** is set to 2 and **'Interval'** to 3, the trail will shrink by 2 points then 3 frames must elapse, and then it will shrink by another 2 points and so on.

You can set a minimum length - again, in terms of the number of points in the trail spline - in the **'Min Length'** setting and add variation to this between different particles.

Very importantly, this operation is non-destructive. Although the trail spline is reduced in length, the trail data is preserved. This implies that the trail could be regrown from the original data, and indeed it can with the **'Expand'** mode.

Expanding trails

This mode of operation will only work if the trail has previously been shrunk with the **'Shrink'** mode. It has no effect on trails that haven't been shrunk. What it does is increase the trail length back to what it was before the shrinking operation started. It won't grow the trail past what it was when it started to shrink.

To see these two modes in operation, see the example file [cb11_trails_trailmod1.c4d](#). This shows the trails shrinking when they enter the field of effect of the first modifier, then expanding again when they enter the second modifier.

Growing trails

With this mode, a trail can be grown longer than the original length as determined by the Trail object. Suppose you have set a **'Trail Length'** of 15 frames in the Trail object. When the particle is affected by this modifier, the trail will grow longer than its original maximum length, without having to be shrunk first. Once again, you control the rate at which it grows with the **'Rate of Change'** and **'Interval'** settings. The actual increase in length is measured in the number of points added to the trail spline, and the maximum extra number of points is controlled by the **'Max Increase'** setting, with variation between particles if desired.

Fairly obviously, this mode has no effect if the **'Full Scene Trail'** switch is turned on in the Trail object since this switch will cause points always to be added to the end of the Trails during the animation, so growing the trails again by adding more points is not possible.

Pulsed trails

One thing you might want to do is shrink a trail, then expand it again, then shrink it and so on to produce a 'pulsed' effect. You could do this with multiple Trails modifiers, but it would be clunky and not guaranteed to produce good results. This is the purpose of the **'Pulse'** mode. It works in exactly the same way as **'Shrink'** mode but when the trail has reached its minimum length it is automatically expanded to its original length again, then the process is repeated continuously.

There is an example file in the download archive, [cb11_trails_trailsmod2.c4d](#), which demonstrates this mode.

11.5.2 The Trails Deformer

Calling this a ‘trails deformer’ is a slight misnomer. Although originally written for the purpose of deforming the trails, it can in fact deform any spline. As with any deformer, it is used by making it a child object of the object to be deformed.

The deformer lets you deform the splines along any or all of the X, Y or Z axes. Very importantly, when used on trails the deformation occurs relative to the particle direction. To see what this means, take a look at the example file [cb11_trails_traildef.1.c4d](#). This uses a Direction modifier to change the particle direction from the positive Z axis to the negative X axis, a 90 degree change in heading. The Trails Deformer is set to deform the trails along the X axis. You can see that, even though the direction changes, the actual deformation occurs relative to whatever the particle direction is in any given frame.

To deform the trails (or other splines) you can use a mathematical formula, or a shader, or both. Each frame, the deformer calculates a value from the formula using the variables and functions in it. That is then multiplied by the **‘Amplitude’** value from the deformer interface to give the actual amount of displacement. This is then applied to each point in the trail or other spline. You can use a formula in any or all of the three axes, and you can use a different formula for each axis if you wish. Note that an empty formula field for any axis returns zero for that axis, so an empty field means no displacement will occur along that axis.

The formula itself can be very simple or quite complex. For example, if you inserted ‘10’ into the ‘Y-Axis Formula’ field, each point in the spline would be displaced by 10 scene units on the Y axis. Much more interesting results are obtained by using various functions and other variables. The default settings generate an animated spline wave along the Y axis.

There are numerous mathematical functions and constants (such as pi) which you can use in the formula: these are listed in the Cinema 4D help files. To find the list, search for ‘formula’ in the help system then look for ‘Formula’ in the search results (in earlier versions of C4D it you may need to look for ‘Formula (Appendix)’).

Additional variables

The Trails Deformer has added some extra variables to the ones available in C4D itself. These are listed in Table 11.1.

It’s easy to make errors when entering a formula (my usual one is to omit a parenthesis somewhere!). Any error will cause Cinema to return the value zero, so there won’t be any displacement. Also, note that all the trigonometric functions in the formula expect degrees, not radians; there doesn’t appear to be any inbuilt function to convert radians to degrees, unfortunately.

Wavelength setting

In addition to **‘Amplitude’** there is also a **‘Wavelength’** setting. This has no effect unless the formula used contains the ‘u’ variable, which is calculated from the wavelength. You can use the wavelength value in the formula directly as the ‘w’ variable, but then all it does is to act as a constant value, which isn’t very useful (though it can be keyframed, so might have a use in some cases).

Using shaders

A shader can also be used to displace the spline points, and again, you can use shaders on any or all of the axes, with different shaders on each axis if desired.

To see how this works, in the example file [cb11_trails_traildef.1.c4d](#), go to the **‘Shader’** quicktab. You will see there is a ‘Noise’ shader in the **‘Shader X’** field. To see this working, change the **‘Mix X’** value to 100%; this will force all the displacement to come from the shader and the formula will be ignored.

The Noise shader causes a random displacement along the X axis, the actual amount of displacement being controlled by the **‘Displacement X’** value. Some shaders give little or no effect in this system, while others are much more interesting. The ‘Noise’ shader is always good value; try the other noise types to see what you get (‘Pezo’ or ‘Sema’ give nice results).

Mixing formula and shader displacement

You can mix the results from these two methods. To do this, use the **‘Mix X’** (or **‘Mix Y’** or **‘Mix Z’**) fields in the **‘Shader’** tab. If you set these values to 0% (the default) all the displacement comes from the formula and the shader, if present, is ignored; if set to 100%, it all comes from the shader. You can vary the proportion each method contributes by changing the mix value.

Variable	Value
x, y, or z	The corresponding coordinate of each point in the spline, relative to the position of the spline object.
t	The current scene time (in seconds) - using this enables the deformation to be animated, as shown by the default formula in the deformer.
w	The 'Wavelength' value from the deformer.
a	The 'Amplitude' value from the deformer.
r	Each point along the spline will be located at some percentage along from the start of the spline. This value is given in the 'r' variable, in the range 0 to 1. The first and last points will always have the same values (0 for the first point, 1 for the last point) but the values for other points may change if the spline length changes with time.
u	<p>This is a special value which uses the 'Wavelength' value from the deformer. Mathematically, it is defined as:</p> $u = p/w - \text{trunc}(p/w)$ <p>Where 'p' is the distance in scene units for a given point from the start of the spline and 'w' is the 'Wavelength' value. In other words, if a spline point happens to be located 440 units from the start of the spline, and 'w' has a value of 200, the 'u' value would be 440/200 (=2.2) minus the value to the left of the decimal point, which is 2. The final value of 'u' for that point is therefore 0.2.</p> <p>You can see that for every point 'u' will always be in the range 0 to 1, no matter how long the spline is. This is important because different points in the spline may have the same 'u' value - and that gives us repeating patterns along the length of the spline. The default formula in the deformer uses this to generate a sine wave. You can see what the 'u' value does a little more clearly if you edit the formula to this:</p> $\sin(u * 360)$ <p>This removes the time variable 't' so that the sine wave does not animate.</p>

Table 11.1. Additional variables for the Trails Deformer

Controlling displacement with the Spline quicktab

This quicktab lets you control the displacement effect along the length of the spline. It does this by sampling the **'Over Length'** spline at a position determined by the distance of a vertex along the trail spline from the start. The value obtained is then multiplied by the displacement from the formula and/or shader controls. If the **'Over Length'** spline value is zero, there will be no displacement; if it is 1 (the maximum) the maximum displacement is applied.

An example makes this clearer. Figure 11.7 shows the **'Over Length'** spline used:

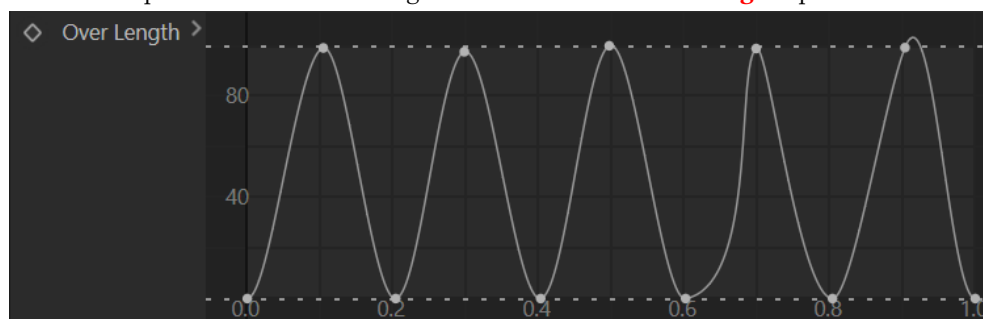


Figure 11.7. Example spline used in the Trails Deformer

Which produces this effect on the trail spline:

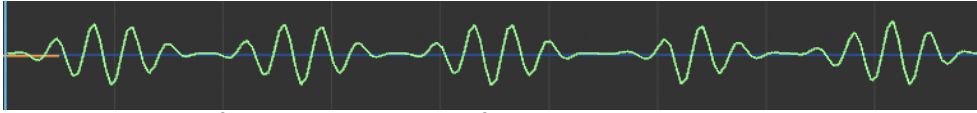


Figure 11.8. Trail deformed using the spline from Figure 11.7

There is an example file in the download archive, [cb11_trails_traildef_2.c4d](#), which reproduces the above effect.

The deformer has two modes. In the first, where **'Spline Mode'** is set to **'Spline Length'** the sampling point of the **'Over Length'** spline is relative to the position of the trail vertex along the spline. For example, if the trail is 100 units long and a vertex is 50 units from the start of the trail, the **'Over Length'** spline will be sampled at 50% along its length for that point. But if the trail grows and is now 200 units long, for the vertex at 50 units from the start the **'Over Length'** spline will now be sampled at 25% along its length.

The second mode is **'Fixed Length'**. Here, you specify a value in the **'Length'** field and for each vertex in the trail the **'Over Length'** spline will be sampled at the position relative to the fixed length value. For example, suppose the **'Length'** value is set to 900 units. For a vertex at 90 units from the start of the trail, **'Over Length'** will always be sampled at 10% along its length ($90/900 = 0.1$). It doesn't matter how much the trail grows, the sampling point will never change.

At first site these modes might not look very different. It is true in the case of the example file that with **'Length'** set to 900 units, the two modes produce virtually identical results on the final frame of the scene. But in that example file, play the animation from the start using one mode then the other. You will immediately see the difference the fixed sample point makes in **'Fixed Length'** mode.

Chapter 12: Questions

Currently there is no way to control X-Particles with a nodal interface. It is possible to control it extensively with Python, but this requires a knowledge of scripting which not all users have or have time to learn.

When originally designing XP the decision was made to add an object-based system of control so that scripting or nodes were not required. This is what the Question and Action objects are for.

↓ The scene file(s) for this chapter can be found in the archive [examples_ch12.zip](#) on the book's website. Click here to [download the archive](#).

12.1 Basic principles

A Question tests whether some item of data meets the criteria you specify. If it does meet the criteria, the question is said to have passed; if it does not, the question has failed. If the test is passed, the Question object must then be able to carry out some kind of operation (or there isn't much point in testing the data in the first place). From this you can see that the only three things you need to consider for a simple question are:

- what is to be tested
- what are the criteria for passing the test
- what operation is carried out on passing the test

This isn't quite everything because in some cases you might want to test two or more criteria at the same time and you also need to consider when the question is tested.

12.1.1 Adding a Question

The only place you can create a Question object is in the emitter's **'Questions'** tab. It isn't possible to create one from the XP menu or anywhere else. To create a Question, click the **'Add Question'** button in the **'Questions'** tab of the emitter. This will automatically create a new Question object, add it to the scene, and drop it into the **'Questions'** list in the **'Questions'** tab. You can then edit the parameters of the Question object in the attribute manager in the usual way.

i Although you can only create a Question object in the emitter, once the new Question is in the object manager you can copy/paste it, alter its parameters and drag and drop it into the **'Questions'** list, if you find that more convenient than clicking the **'Add Question'** button again.

12.1.2 What is to be tested?

This is determined by the **'Choose Question'** section in the object's interface:

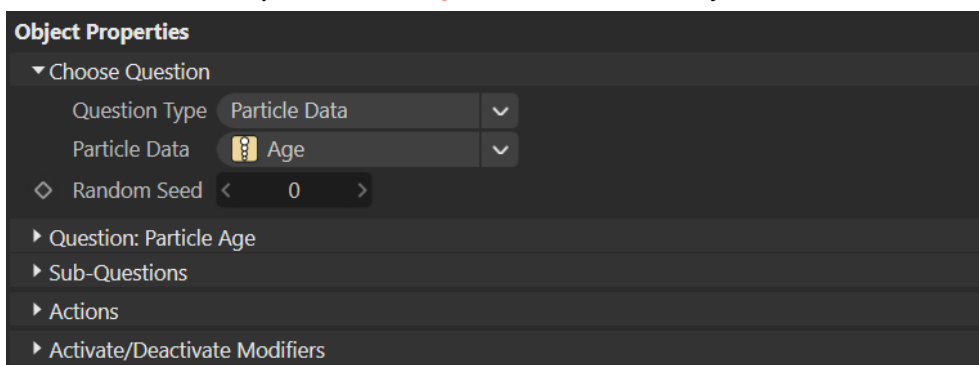


Figure 12.1. Setting the data to be tested

Question Type

The **'Question Type'** menu gives access to several different sets of data to test. These can be data which is part of the basic particle data, extended data such as physical or fluid data, or data which is nothing to do with a particle such as scene time or a random probability. So the first thing is to choose the question type. There are seven types as listed in this table:

Question type	Data tested
Particle Data	Basic particle data such as age, speed, radius, etc. Some extended data is also tested here, such as world speed and Bullet collisions, and any custom data you create can also be tested in this question type.
Particle Position	Various data associated with the position of a particle, such as the distance to an object or the distance to the nearest other particle.
Particle Flags	Within each particle XP maintains a number of flags, which are basically switches which are set if a particular condition occurs. For example, if a particle collides with a scene object which has an XP Collider tag attached, a flag indicating that a collision has occurred will be set in the particle.
Fluid Data	These are data items which are available when a particle is part of a fluid simulation.
Geometry	These questions test aspects of geometry generated from a particle.
Physical Data	Physical data includes such things as mass, temperature, fire. and so on. The emitter's physical data also includes a user value which you can set to anything (rather like an inbuilt custom data item) and that value can be tested in this question type.
Other	A miscellaneous group of data items which don't fit anywhere else, such as the scene time or a random probability factor.

Table 12.1. Question types

Data to test

As you can see from Figure 12.1, there is a second menu labelled '**Particle Data**'. This menu will change its name, and its contents, when the '**Question Type**' is changed. For example, if you change the type to '**Fluid Data**' the second menu will also be labelled '**Fluid Data**' and the entries in that menu will be different. It is from this menu that you select the actual data to test. We will look at this in detail below.

'Random Seed'

This field is a see value for a random number generator, which is used in the '**Random Probability**' question and in some other question types.

12.1.3 What are the criteria for passing the test?

Clearly, the actual criteria will vary with different data to be tested, but most questions have a '**Mode**' menu, many of which have common entries. It is useful to list these here because they crop up repeatedly with many different questions.

Not all questions have these modes; some have some of them but not others, and some have none of them, but where they are present they all work the same way. The common modes are as follows:

Is Greater Than

To pass, the test value must be greater than the parameter you specify in the Question object. For example, in the Particle Data: Speed question, you specify a speed value and the particle's speed must be greater than that to pass, as shown in Figure 12.2.

Is Less Than

The opposite of 'Is Greater Than'. The test value must be less than the specified parameter.

Equals

The test value and the parameter must be equal. Be careful with this mode. It will work without problems if you are testing an integer value such as the particle ID but you will see problems with floating-point numbers if the values are not exactly equal. For example, if you are testing the particle speed and you set the '**Speed**' parameter to 200, then the particle speed must match that exactly in order to pass. If the speed is 199.99 or 200.01, the question will not pass. Floating-point numbers can often differ very slightly from the value you think it should be due to the way floating-point calculations in the CPU work.

If you really need to test if a speed or other floating-point values equals the parameter value, don't use 'Equals' as the mode, use the '**Is In Range**' mode (see below) with a very narrow range. In the above example you could set a value of 200 with a range of 0.1. This will cause the question to pass if the speed is anywhere from 199.9 to 200.1 - which should be accurate enough for most purposes.

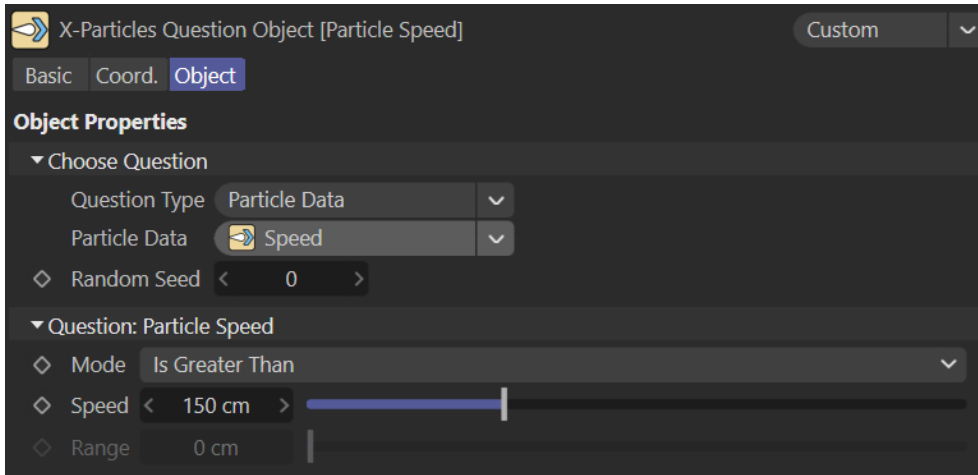


Figure 12.2. Question mode 'Is Greater Than'

Is In Range

This mode lets you specify a range of parameter values. If the test value falls within that range, the question will pass. The range is specified by first entering a base value - for example, if testing the particle radius, you might enter a radius value of 3 units. Then you enter a range value, e.g. 0.5 units, like so:

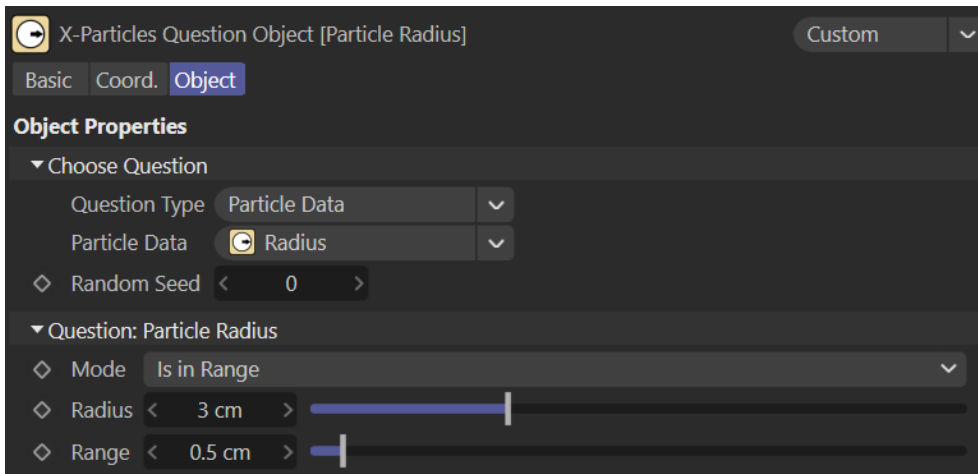


Figure 12.3. Question mode 'Is In Range'

The final range is calculated with the minimum value being the base value minus the range value, and the maximum is the base plus the range. In the above case the range would be 2.5 to 3.5 units. If the particle radius falls within that range, the question will pass.

⚠ There is an issue to be aware of with this mode. If you have a test value which is increasing on a regular basis - for example, the particle age automatically increases each frame, or you might be using a Scale modifier to increment the particle radius each frame - then the question will pass as soon as the test value enters the range. As an example, if you are testing the particle age with this mode with a base value of 50 frames and a range of 10 frames (so giving a range of 40-60 frames) since the age increases each frame the question will pass as soon as the age is 40 frames. This gives the same result as using the **'Equals'** mode. This mode is therefore best used for floating-point values where **'Equals'** can't be relied on (see **'Equals'** mode above for the reason) or for some parameter which might be continually increasing and decreasing with time, such as speed, radius, distance to an object and so on.

Another way around this problem when testing particle age is to use the **'Fixed Within Range'** mode, which is discussed below in the **'Age'** question.

Is Not In Range

With this mode you specify a base and range value as with 'Is In Range' but the question will only pass if the test value is not within that range.

In addition to the ‘common’ modes, many questions have specific modes of their own, depending on the value being tested. These are discussed under the individual questions.

12.2 Question types

12.2.1 Particle Data questions

Age

Tests the particle age. There are two modes specific to this question. The first is **‘Is Within Time of Death’**. Since each particle has a fixed lifespan, knowing its age we can calculate how much time must elapse before the particle dies, assuming it isn’t killed earlier by some other means. This question will pass if the particle has the same or less time before death than is specified in the **‘Age’** parameter.

The other unique mode is **‘Fixed Within Range’**. We saw that the **‘Is In Range’** mode has a problem in that a particle may pass the age question as soon as it enters the range. What you might prefer is that each particle will pass at some random age which falls somewhere within that range. If you select this mode, and specify a base and range value, every particle will be given its own randomly-generated fixed value within the range and for each particle the question will only pass once that age value is reached.

Color

Tests the particle colour. There is an additional menu where you specify which of the RGB components you want to test; the default is to test all three, which in effect is a greyscale value which tests the colour brightness.

Direction Change

This tests whether a particle has changed direction, and if so, by how much. The **‘Direction Axis’** menu lets you specify which axis the change must occur on. One option is **‘Any’** which means that any change in direction no matter how small will cause the question to pass. For any other axis you can specify by how much the direction should change, in degrees. There are only two modes: **‘Less Than’** will pass the question if the direction change is less than the specified value. **‘Equal To Or Greater Than’** will pass if the direction change is, well, equal to or greater than the **‘Direction Change’** value.

There is an important caveat with this question. What it is actually testing is whether the direction has changed since the previous frame. For that, the previous frame’s direction must be stored in the particle as well as the current direction, which isn’t normally the case. To store that value, you must turn on the **‘World Speed’** switch in the emitter’s **‘Extended Data’** tab. If that is not done, the question will always fail to pass.

Finally, you can set a minimum age and only after the particle has reached that age will the question pass.

Group

With this question you can drag and drop one or more groups into the **‘Groups’** list. There are two modes, which are very simple: either the particle being tested is in one of the groups in the list, or it is not in any of those groups.

Metagroup

This is identical to the ‘Group’ question but for metagroups instead. The only difference is that you can only specify one metagroup, not a list of them.

Particle ID

Tests the unique ID value of a particle. With this you can test one specific particle if needed, or you can give a range of possible values. There are two additional modes: with ‘Even-Numbered Particles’ the question will pass if the particle ID is an even number; ‘Odd-Numbered Particles’ will pass for any particle whose particle ID is an odd number. Note that with both these modes the **‘Particle ID’** and **‘Range’** parameters are not available.

Radius

Tests the particle radius. There are no unique modes to this question.

Rotation

Tests the rotation of the particle. This doesn't mean that the particle is actively spinning on any axis. It tests whether the particle has rotated away from the default rotation of zero degrees on all three axes. You can specify the axis to test and the amount of rotation.

Scale

Tests the particle scale. You can specify which axis to test and the scale parameter in the **'Component Values'** field. Note that if you choose the axis to be **'X Only'** (for example) you can still change the Y and Z component values but they are ignored; only the X component would be tested.

Speed

Tests the particle speed. There are no unique modes to this question. You should be aware that what is being tested is the particle speed as held in its internal data. This may not be the same as the actual speed in the 3D world. For why this is the case, see the emitter section 4.3.3 'Stick Particle to Source Object'.

World Speed

Tests the actual speed of the particle in the 3D world, whether that is driven by the emitter or because the particle is stuck to a moving object. To test this value, you must first turn on the **'World Speed'** switch in the emitter's **'Extended Data'** tab.

Time in Group

When a particle is created, it is always added to a particle group. However, that group can change by using objects such as the Change Group modifier. What if you would like to change a group after a particle has been in a group for a certain time? You can do that with this question. What it does is test the time that the particle has been in its current group - you don't need to specify a particular group. The usual modes are available and if the question passes you could use it to change the particle to another group, for example.

Custom Data

This question lets you test a custom item of data you have added to the particle. Custom data is fully discussed in the emitter section 5.5 and there is an example of using a question to test it in section 5.5.3 'Testing the data'. You need to specify which custom data item you are testing and, very importantly, what kind of data it is. For example, if the data item is actually a vector and you leave **'Data Type'** at the default **'Integer'** setting the question will almost certainly fail or at the very least may give unexpected results.

Collision (Bullet)

With this question you can test if a particle has collided with an object in the scene when using X-Particle's implementation of the Bullet dynamics system. To use this, you will need to set up objects with Bullet collider tags and drag and drop them into the Question object's **'Collision Objects'** list. Finally, you must add a Bullet rigid body tag to the emitter.

When the particle collides with one of the objects in the list the question will pass. Alternatively, you can opt to pass the question if a collision takes place with an object that is not in the list. The example file [ch12_questions_bullet.c4d](#) in the download archive demonstrates this working. There are two objects in the scene, a Cube and a Sphere, plus two Question objects which test for Bullet collisions. The Question objects have **'Collision Objects'** lists which both contain only the Sphere object. In the first question, **'Mode'** is set to **'Collided With Object NOT In List'**. If the particles hit the Cube (not all do) then since that object is not in the Question's object list, they turn red. If any particle, regardless of whether it hit the Cube or not, goes on to collide with the Sphere, it will turn green since the second question's mode is set to **'Collided With Object In List'**.

i Note that with Bullet the particles are also objects and will collide with each other; you don't have to specify this separately. In the example file try increasing the emission rate to 100 rather than 10. You will now see that some of the particles turn red even before they collide with the Cube. This happens because the particles are colliding with other particles and since they are not in the object list of the first question, they will turn red.

12.2.2 Particle Position questions

Distance Along Spline

To use this question, a particle must be affected by a Follow Spline modifier, which causes a particle to move along a spline. You don't need to specify the spline in the question because that is the spline used by the modifier.

First there are two simple tests in the **'Parameter'** menu - **'Reached End of Spline'** and **'Reached Start of Spline'**. The question passes once the particle has reached the end (or start if moving 'backwards' along the spline) of the spline.

See the example file [ch12_questions_along_spline_1.c4d](#) to see how this works. The question tests if the particle has reached the end of the spline and when it does, it is turned red. This particular spline is a closed spline so the start and end point are in the same position. However, what is being tested is how far the particle has moved along the spline, and even though a particle might look to be at the start when it is really at the end, it has actually moved along 100% of the spline's length - so cannot be at the start. You can try this by changing **'Parameter'** to **'Reached Start of Spline'**. You will then see that the particles do not change colour, so the question has not passed.

i If you want to test the **'Reached Start of Spline'** parameter, you will have to make some changes in the example file. Specifically, you should:

- in the Follow Spline modifier, change 'Starting Mode' to 'Position Along Spline'
- in the modifier, change 'Position' to 100%
- and also in the modifier change 'Move Direction' to 'Backwards'
- finally, in the question change 'Parameter' to 'Reached Start of Spline'

Now you will see the particles move in the opposite direction, from the end to the start, and turn colour when they reach the start of the spline.

The next parameter is **'Distance Along Spline'**. To see this, reload the example file and change **'Parameter'** to **'Distance Along Spline'**. With the default settings you should see the particles turn red at the halfway point along the spline.

A You should be aware that the **'Distance'** setting is always measured from the start of the spline to the end, regardless of the modifier 'Move Direction' setting. To see what this means, with the example file still set to use 'Distance Along Spline' in the question, change the **'Distance'** setting to 25%. If you play the scene, the particles turn red at a quarter of the distance along the spline, as expected. Now, in the modifier, change 'Move Direction' to 'Backwards'. Now what happens is that the particles change colour the moment they arrive at the spline. Why?

With backwards movement, as soon as the particles start to follow the spline they are at the end of the spline - that is, 100% of the distance along it. The question tests if the distance is greater than 25%, which it is, so the colour change occurs. To fix this, change the **'Mode'** to **'Is Less Than'**. Now the particles will only change colour at 25% of the way along the spline, as expected. If you wanted to have them change when they had actually travelled a quarter of the distance backwards along the spline instead of at the fixed 25% position, you would have to invert the distance. Instead of 25%, set the **'Distance'** to 75% and then they only travel a quarter of the way around before changing colour.

The final parameter is **'Number of Loops'**. The Follow Spline modifier normally stops particles following the spline when they reach the end, but it can be made to work in a continuous loop. To see this, try the example file [ch12_questions_along_spline_2.c4d](#). The modifier now loops, and the question will pass when the particles have made more than two loops. If you play the file and keep count of the number of loops the particles make, you can see that they change colour as soon as they complete three loops.

Distance to Camera

This tests the distance of a particle to the current active render camera (not the viewport camera since these can be different). There are no modes other than the common ones.

Distance to Object

This question tests the distance of a particle to a specified object. But what point on the object is used to measure the distance? You have two options in the **'Distance To'** menu. The default is **'Object Position'** which uses the position of the object's axis.

The alternative is '**Nearest Surface Point**' which attempts to find the nearest point to the particle on the surface of the object. The key word here is 'attempts'. In practice, the question first finds the nearest polygon (which is straightforward) and then tests random points on the surface to find the closest point. Clearly, the more random points which are tested the more accurate the eventual 'closest point' is likely to be; this is the reason for the '**Accuracy**' setting. This is actually the number of random points tested. It turns out that if the nearest polygon is a small one, the number of random points tested can be kept quite low, but if it is large, the '**Accuracy**' value may need to be increased. Likewise, if the distance is short more points need to be tested because then a lack of accuracy becomes really noticeable. You will need to adjust the '**Accuracy**' setting depending on the object mesh and how accurate you need it to be. Of course, the greater the number of points tested the slower the process will be.

Other than the '**Distance To**' setting the available modes are the common ones and you simply need to specify the distance, and range if necessary, that you want to use.

The example file *cb12_questions_distance_to_object.c4d* shows this question in action. Try changing the '**Distance To**' to '**Nearest Surface Point**' then use different '**Accuracy**' settings to see the effect on the animation playback speed.

Distance Travelled

As the name suggests, this question tests the distance the particle has travelled in the 3D world. However, there is a slight complication. A particle can be moved in two ways. It may move as determined by its own velocity, or it may move because it is stuck to an object which is moving. Unless the particle is stuck the first method will be what you need - you can use the second method anyway, but it adds additional data to the particle and may be slower to calculate. The method used is controlled by the '**Recorded Distance**' menu.

When this is set to '**Local to Particle**' the particle's own velocity is used to calculate the distance but if the particle is stuck to an object, whether or not the object is moving, the distance travelled will be zero since the particle is not moving under its own steam. The alternative setting is '**World**' and then all particle movement is taken into account, whether it is moving by itself or because it is stuck to an object. To use this method you must turn on the '**World Speed**' switch in the emitter's '**Extended Data**' tab.

The other parameters in the question are the common question modes and the distance/range settings.

Illumination

This question tests if a particle is in the field of illumination of a Light object. Currently it requires that the light is a spot or omni light; no other types of light will work. The mode choices are very simple: either the particle is in the illumination field ('**Illuminated**') or it is not ('**Not Illuminated**'). You must also supply the Light object to be tested.

The example file *cb12_questions_illum.c4d* shows a very simple example with a spotlight. There are two Question objects: one tests if a particle is in the illumination field and if so turns the particle colour to green, the other tests if the particle is not in the illumination field and if so turns it red. You can see if you play the scene that the particles start off red (because they aren't in the illumination field), turn green when they enter the field, then turn red again when they leave it.

There is an additional parameter named '**Illumination Threshold**'. For the '**Illuminated**' question to pass, the particle must be in an area of the illumination field which gives illumination greater than this value. In the example file this won't make any difference because the cutoff between the inside and outside of the illumination field is immediately from 0% outside to 100% inside the cone of the light. For the 'Non-Illuminated' question, the illumination field must be less than or equal to this value.

There is one other parameter: the '**Use Falloff**' switch. If the light has a falloff, that will be taken into account only if this switch is turned on. The example file *cb12_questions_illum_falloff.c4d* demonstrates this and the use of the '**Illumination Threshold**' parameter. The Light object is an omni light with a linear falloff. Play the scene and note that the particles do not change colour immediately on entering the illumination field. This is determined by the threshold settings in the two Question objects.

In Figure 12.4 the threshold setting for the '**Illuminated**' question is 0% on the left and 45% on the right. You can see that increasing the threshold means that particles furthest away from the Light object are not as illuminated (due to the light falloff) as the closer ones, so for them the question does not pass:

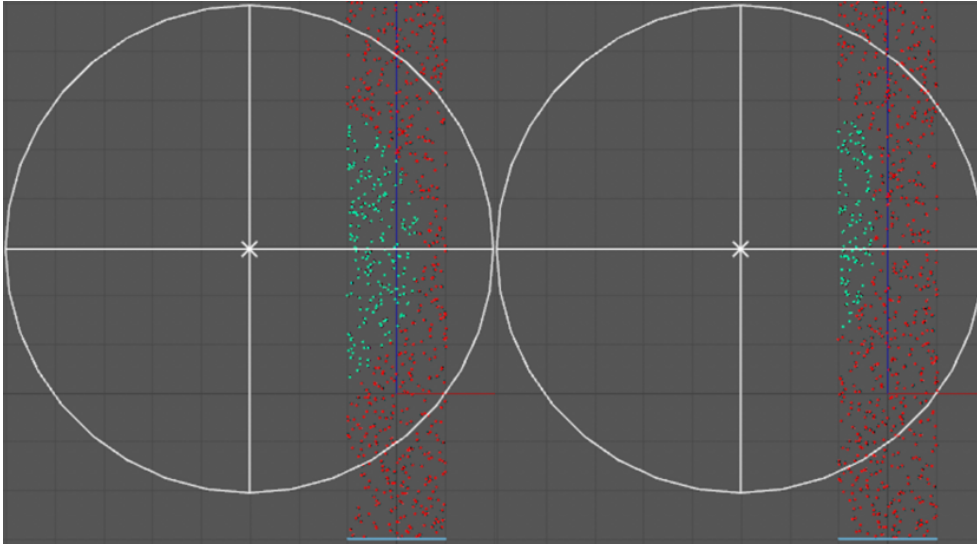


Figure 12.4. Different illumination threshold settings for the 'Illuminated' question mode

In Figure 12.5 the threshold settings for the '**Not Illuminated**' question are 5% on the left and 35% on the right:

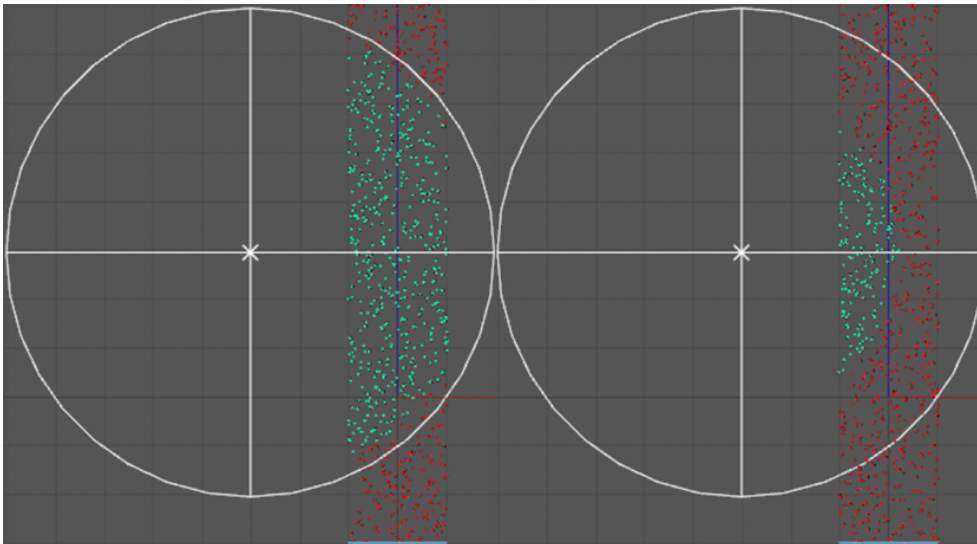


Figure 12.5. Different illumination threshold settings for the 'Not Illuminated' question mode

Remember that in this case for the question to pass and turn the particles red, the illumination must be less than or equal to the threshold value. As that value decreases, the particles which are illuminated less than the threshold become less numerous and the number of red particles will fall.

⚠ Note that with this question the test is whether the particle is within the illumination field of the light, not whether the particle is actually illuminated. If we were to place a plane object between the light and the particles, no light would reach the particles but the question would not be affected because the illumination field has not changed.

In Camera FOV

This question tests whether or not a particle is in the field of view of the current camera in the viewport, including the default editor camera. The only parameter is the option to test if the particle is within the FOV or not. You don't need to specify the camera because the camera currently in use is always used in the question.

Inside Volume

Another simple question which tests whether a particle is inside the volume of an object or not. You must specify the object to use. This doesn't necessarily have to be a closed object such as a Cube or Sphere, but it must enclose or partly enclose a volume of space. In other words, a Plane object won't work. However, if you distort the Plane so that it is no longer two-dimensional, it will start to work.

In Figure 12.6 a Bulge deformer has curved the Plane slightly (you may have to look closely to see it) so that the Plane is no longer a simple 2D object and therefore the question passes for most particles. As the bulge strength is increased more particles will be in the volume at some point and will therefore pass the test:

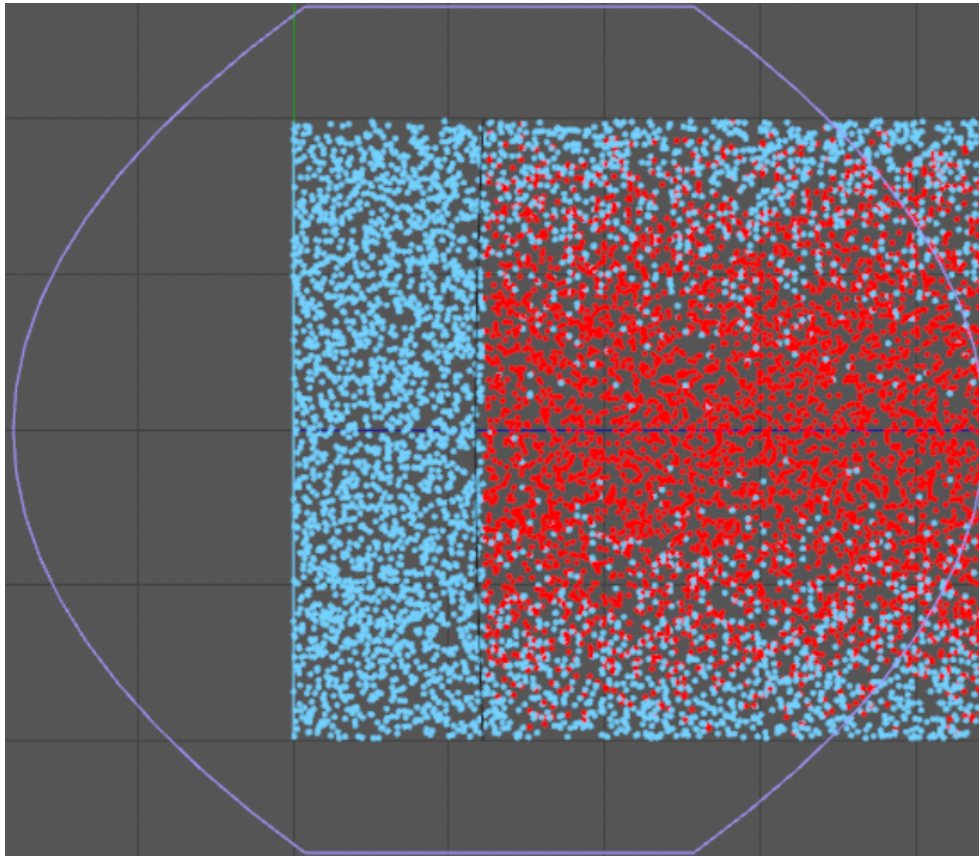


Figure 12.6. Testing if a particle is inside the volume of a deformed Plane object

This scene is one of the example files provided, [ch12_questions_inside_volume.c4d](#). The Bulge deformer is disabled at first and you see that no particles change colour, because the Plane does not enclose a volume. If you turn the Bulge deformer on, you will see the effect shown above. Try altering the Bulge 'Strength' setting and see how this affects how many particles pass the question.

Modifier Falloff

This is a question which tests if a particle is within the field of effect of a modifier's falloff (pre-Cinema 4D R20) or Field object. You must specify a modifier but it can be anything, even if that modifier has no effect on the particle. In the example file [ch12_questions_mod_falloff.c4d](#) I've used a Trails modifier which clearly has no effect on the particle since there is no Trail object in the scene. You can use this basic file to experiment with the modes and parameter values in the Question object.

The **'Mode'** menu has several options. First, you can simply test if the particle is within the falloff/field or outside it. If a particle is outside a falloff/field the value returned by the falloff is always zero and begins to increase towards a maximum of 1.0 as the particle enters the field of effect. You can test this value with the other three options. The first is **'Falloff >'** where to pass, the falloff value must be greater than the value in the **'Low Falloff Value'** parameter. In this screenshot using a spherical falloff (from R19), the falloff value will be 100% inside the inner sphere then will decrease to zero when moving towards the outer edge of the falloff. The **'Low Falloff Value'** is set to 60% and you can see in Figure 12.7 that the particles change colour to green when they get close to, but not yet inside, the inner sphere.

⚠ Note that in this question if you set the **'Low Falloff Value'** to 100% the question will never pass since the falloff value cannot exceed 100%. You can, however, set it to 99.99% and it will work as expected.

The opposite test is **'Falloff <'**. To pass, the falloff value must be less than the **'High Falloff Value'** parameter. As with the previous mode, if you set this value to 0% it will never pass since the falloff value cannot be less than zero. Finally, there is the **'In Range'** option where the falloff value must fall between the low and high parameter values.

The only other parameter is **'Weighting'**. This is a random probability that, even if the falloff value criteria are met, the question

will actually pass. If it is set to 100% it will always pass if the falloff criteria are satisfied; if the weighting is 0%, it will never pass. If your aim is to have some particles pass the question but not others, you should set this to a low value, such as less than 5%. The reason is that the question, if it does not pass, will be retested each frame and eventually random probability indicates that a particle will pass the test. If this value is set too high, all or most particles may pass within a few frames.

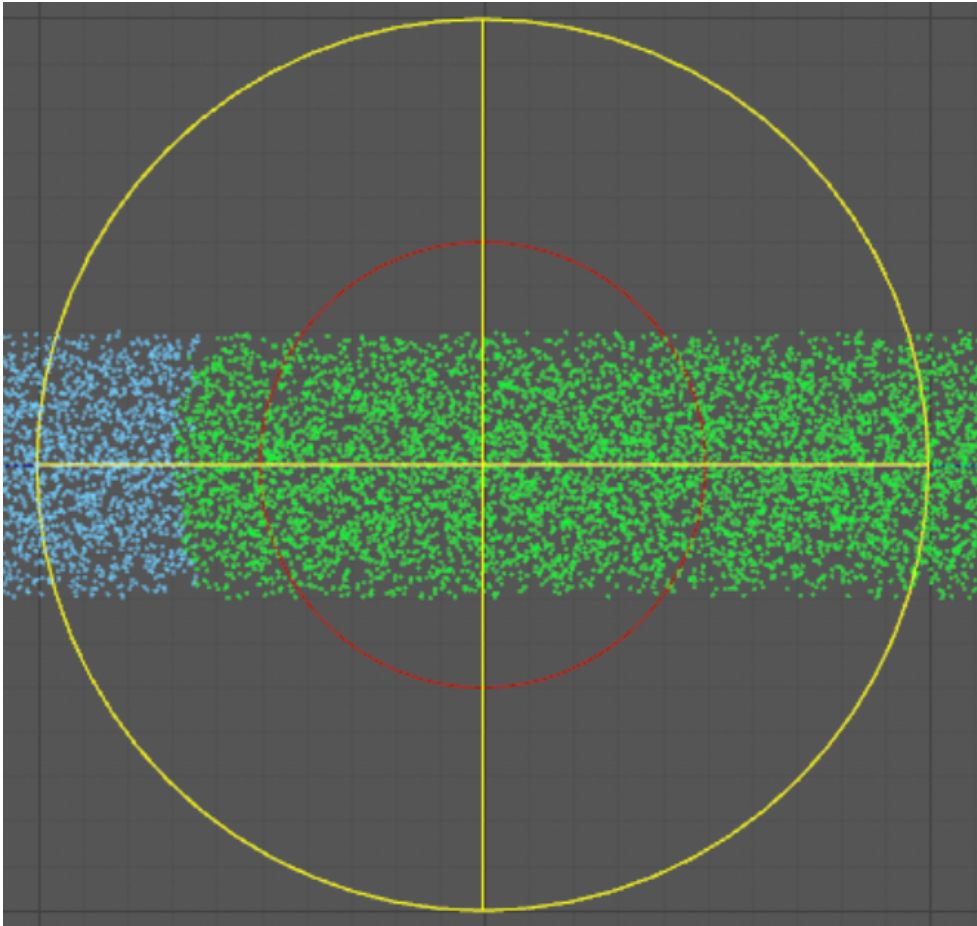


Figure 12.7. Testing if a particle is within the falloff of a particle modifier

Nearest Particle

This question tests the distance to the nearest other particle, or to the furthest other particle. If this question is tested repeatedly during the animation the result may change from one frame to the next, depending on which particle is nearest to (or furthest away from) the one being tested. Other than specifying which parameter is to be tested, you can choose between two of the common modes and then set a distance parameter.

Particle Position

With this question you test perhaps the most fundamental of all particle properties: where it is in the 3D world. First, you choose the **'Position Space'**. This can be the absolute position in the 3D world with the **'World'** setting, or relative to an object you specify by dragging it into the **'Object'** field.

With that done, you can now choose the axis along to measure the distance to the particle. If the **'Position Space'** setting is **'World'** and the X-axis is chosen, what you are testing is the distance along the X-axis from the particle to the centre of the 3D world. If **'Position Space'** is **'Object'** and the Y-axis is selected, you are testing the distance along the Y-axis from the particle to the object's axis.

Finally, you can set one of the common test modes and the distance to test. The example file [cb12_questions_particle_pos.c4d](#) shows how this question works and demonstrates an important caveat. The question tests the distance from a Sphere object along the Y axis, the **'Distance'** setting is 75 units, and the mode is **'Greater Than'**. When the question passes, the particles will turn red. Figure 12.8 shows the result after 75 frames:

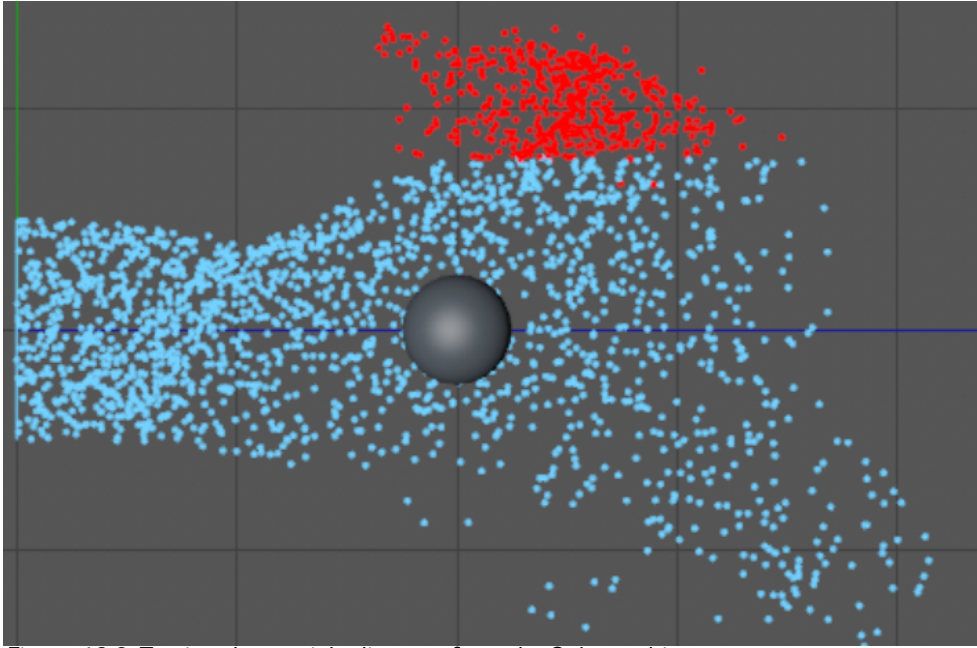


Figure 12.8. Testing the particle distance from the Sphere object

As expected, when the particle are 75 units above the Sphere's axis position, they turn red. But you might expect those which are 75 units or more below the Sphere's axis also to turn red - after all, they are more than 75 units away. But they don't do so. The reason is that it is the actual particle position being tested, not simply its distance from the object. If you wanted to test if particles were 75 units or more below the Sphere, you would use '**Less Than**' as the mode and -75 units as the distance.

Vortex Radius

The Vortex modifier sets particles spinning as though caught in a tornado. The tornado has a radius which can be increased over time. See the example file [ch12_questions_vortex_radius.c4d](#) which is a very simple example of the Vortex modifier. Then enable the Question object in the object manager and you will see particle turn red when they exceed a radius of 50 units (relative to the centre of the particle vortex) is exceeded.

The only parameters are the common modes and the settings for the radius (and range when required) to be tested.

! This question requires that the particles are affected by a Vortex modifier, which adds the necessary data to the particle. If there is no Vortex modifier affecting the particle, the question will always fail.

12.2.3 Particle Flags

X-Particles sets and clears a number of internal switches or flags in each particle depending on various events which happen to them. Some of these can be tested in this question type (the others are mostly for internal use and testing in a question would not be useful).

Most of the individual questions of this type don't have any parameters. They are set when an event occurs and may be cleared later, but the only test is whether the flag is set. If it is, the question will pass; if it is not, it will fail.

Note that this does mean that it is not possible to test if a flag is not set - only if it is. There is a way around this, however, using custom data where you are in effect creating your own flags. See the example file [ch12_questions_istuck.c4d](#), which is explained below in the '**Stuck to Source Object**' question, so you can see how this is done.

Particle (Collided) Object

This flag is set if a particle collided with a scene object (not another particle) which has a Collider tag attached. Note this does not work for Bullet collisions, which have their own question, as discussed above. The question will pass if the flag is set.

One important point to remember is that the flag is set when the collision occurs but it is cleared in the same frame after all questions have been processed. This means that you must test the flag in the frame the collision occurs - you can't wait for some other

event several frames later and then test this flag as well. If you need to do that, by using custom data you can set your own flag and test it later. This is shown in the example file [ch12_questions_flag_collided_persist.c4d](#) and deserves a little more explanation.

In this scene the emitter has an item of custom data named 'Has Collided', which is an integer with the initial value of zero indicating no collision. When the particles collide with the Cube, an Action is called which sets the value of the custom data to 1. We'll look at how Actions are triggered like this in Chapter 13.

There are two Question objects: the first tests the particle age and passes if the age is greater than 90 frames. The other tests if the custom data item named 'Has Collided' is set to 1; if it is, that can only be if a collision took place at some point, which might have been many frames previously. In this case the actual particle flag is not being tested because that will have been cleared by the emitter, but we have effectively persisted the flag in the form of a custom data item. The result is that the particles turn red only if the age is greater than 90 frames and a collision with an object has taken place at some point.

Particle-Particle Collision

This works in the same way as 'Particle Collided (Object)' but for collisions with another particle. To use this question particle-particle collisions must be enable using an xpPPCollisions object. How this question works is shown in the example file [ch12_questions_ppcollisions.c4d](#).

Has Exploded

This question requires an Explode modifier to affect the particles. The question will pass if the particle has been 'exploded' by the modifier.

Particle Stick Question

This question requires a Cover/Fill modifier. With that modifier, when the particle arrives at its target point, a flag is set in the particle, which is tested by this question. The flag remains set as long as the particle is stuck to the target, but if it is released the flag will be cleared. The example file [ch12_questions_particle_stick.c4d](#) shows this working.

Snapped (Inheritance Mod.)

This question also requires a modifier, the Inheritance modifier. Briefly, this modifier forces the particles of one emitter to inherit the properties of corresponding particles from another emitter. One such property is the particle position. When that property is used, the modifier can snap the particles to the position of their targets, and that will set a flag that can be tested by this question.

In the example file [ch12_questions_inherit_flag.c4d](#), you see the 10 yellow particles moving to the position and rotation of the target particles. When they finally snap into position, this flag is set and the particle change from a yellow outline to a filled red shape.

Stuck to Source Object

A particle emitted from an object can be stuck to it using the '**Stick Particle to Source Object**' switch in the emitter. This will set a flag in the particle which can be tested with this question, and the question will pass as long as the particle is stuck to its source object.

It might be more useful if you could test when a particle is no longer stuck. Unfortunately the Question system at present doesn't allow actions to be taken if a flag is not set, but there is a workaround using custom data. The example file [ch12_questions_istuck.c4d](#) shows this working; the particles turn red when released from their source object.

The way it works is this. We want to carry out some action when a particle is no longer stuck to the source object, but we can't test the flag directly because the question will only pass if the particle is stuck to its source, not when it is unstuck. In this case, we first create an item of custom data in the emitter and set it up as shown in Figure 12.9.

Then a Trigger Action modifier (whose sole purpose is to trigger Action objects) is animated to pass across the Sphere to which the particles are stuck. As it does so, it will trigger actions to do two things: unstick the particle and change the custom data value from 1 to zero. In the Question object, the custom data is tested and if it is zero, indicating that the particle is no longer stuck, the colour is changed to red. You can use this technique for any of the flags where you want to test if the flag is not set, rather than if it is.

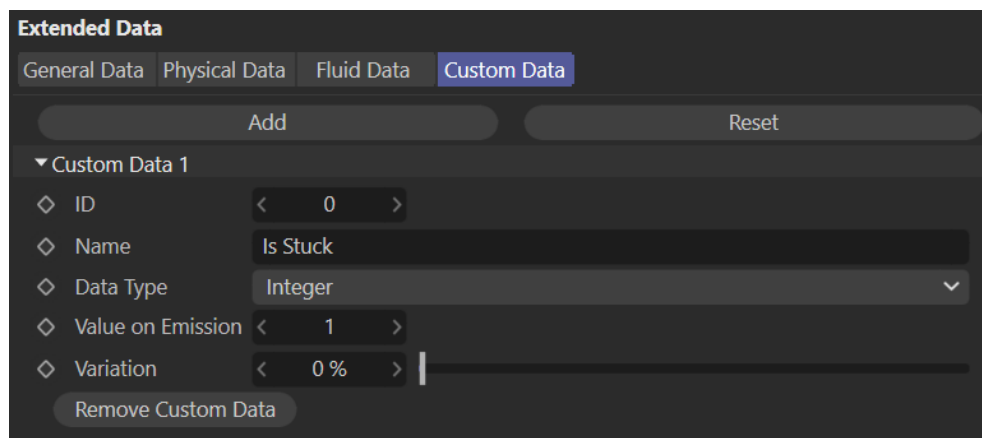


Figure 12.9. Adding custom data to test if a particle flag is not set

Has Connections

This is one of the two particle flags questions with additional parameters. The question tests if the particle is connected to an object or another particle. Such connections can be made by the Constraint object, or on collision with an object, or on collision with another particle. The only parameter is the type of connection you are interested in. If such a connection exists for the particle being tested, the question will pass.

Particle Frozen

X-Particles offers several ways to freeze a particle: the Freeze modifier, the Freeze Action or the Freeze tag. You can also opt to freeze any combination of movement, rotation, or change in scale. Freezing the particle will set flags depending on which parameter(s) have been frozen.

! This question will not work if the particle has been frozen with a Freeze tag. The tag is one of the oldest parts of X-Particles and it is recommended that the corresponding modifier or action is used instead.

You can turn on any of the three switches in the question. If **'Operation'** is set to **'Match Any'** then the question will pass if at least one of these switches is turned on and any one or more of the 'freeze flags' in the particle is set. The alternative is **'Match Exact'** so that the question will only pass if the switches that are turned on match the flags exactly. For example, if the **'Frozen Movement'** switch is on, and only particle movement (but not rotation or scale change) is frozen, the question will pass. But if the movement and spin switches are on, but only particle movement is frozen, the question will fail.

12.2.4 Fluid Data

These three questions all test fluid data stored on the particle. They all require either a PBD Fluid object or a Fluid FX object to work and all use the common question modes with appropriate parameters.

Fluid Density

Tests the fluid density at the particle's position; requires a PBD Fluid object. The parameters are the common parameters described at the start of the Questions section.

Fluid Surface

Requires a Fluid FX object. This question is testing whether a particle is close to the surface of the fluid simulation, but the **'Surface Test'** parameter is not an actual distance. Instead it is a relative value with higher values meaning that the particle is closer to the surface. See the example file [ch12_questions_fluid_surface.c4d](#). This is a simple Fluid FX setup and particles are tested after they are 90 frames old (to allow for some stabilisation). You can see that the yellow particles are concentrated mainly at the surface, but bear in mind that this is a 3D simulation so 'surface' doesn't mean only particles at the top of the fluid on the Y axis, which is why you also see yellow particles on the sides and the bottom of the simulation. Try altering the **'Surface Test'** parameter and the value in the 'Particle Age' question to see the results.

Granularity

This also requires a Fluid FX object. The ‘granularity’ is a measure of how many particles are near to the one being tested. In a way this is the opposite of the ‘Fluid Surface’ question since the highest granularity is likely to be found in the centre of the fluid.

12.2.4 Geometry

The questions in this section all relate to some kind of geometry linked to a particle.

Branching

All these questions require that a Branch modifier is affecting the particles. The Branch modifier stores a large amount of data on the particle, some of which can be usefully tested in this question. There are four types of question you can choose from within the ‘Branching’ question. All use the common question modes and for each one you specify the parameter to test against. The four types are as follows:

- **Branch Length:** the length of the branch associated with the particle being tested; the parameter is the length in terms of the number of frames for which the branch has grown. Note that this is not a time setting but an integer value.
- **Max Length (in Frames):** the maximum length of the branch as set in the Branch modifier and Sub-Branch objects. Again, this is an integer setting, not a time.
- **Branch Level:** any branch may generate sub-branches and they in turn can generate sub-branches of their own. Each time one or more sub-branches are generated from a branch the level increases. This question lets you test if a branch is on a particular level. The example file [ch12_questions_branch_level.c4d](#) shows this working. There are three branch levels in this scene: the initial single ‘stem’, which is level 0, the branches from the stem (level 1) and sub-branches from these branches (level 2). There are three Branching questions, one for each level, and each one alters the appearance of the particles associated with that level.
- **Number of Branches:** the number of branches which are branches from the branch controlled by the particle being tested. In other words, if a particle is on branch level 1, and has 12 sub-branches, the question can be used to test if this number (12) is greater than, equal to, or less than the ‘**Value to Test**’ parameter. Note that if any of those sub-branches themselves have sub-branches, they are not included in the test – only those which are direct branches from the particle tested are used.

Generated Object

This question tests the object associated with the particle and being generated from a Generator or Sprite object. If you select ‘**Generator**’ from the ‘**Object Type**’ menu, you can test the index number of the object which is generated for that particle. Recall that a Generator can have multiple child objects and can generate any one of them for each particle. The first child object will have an index of 1, the second 2, and so on. You specify which index value to test in the ‘**Object Index**’ parameter and depending on the ‘**Mode**’ parameter, the question either tests whether an object with that index is, or is not, associated with the particle.

If you select ‘**Sprite**’ from the ‘**Object Type**’ menu, the test is then which type of sprite is being generated. Instead of an index number you select the type of sprite from the ‘**Sprite Type**’ menu. Again, the question tests whether the sprite is, or is not, of that type.

Morph Value

Morphing is discussed in detail in the Generator object section. Once morphing commences, the object associated with the particle has a morph value ranging from zero to the maximum value for the particle. This question simply tests the morph value.

The mode ‘**Morph at Max**’ is included because you may not know what the maximum morph value is for a particle. For example in the Generator if the ‘**Morph Max**’ is set to 50% with zero variation, and you wanted to test if the particle morph value had reached its maximum, you could simply test if the morph value was equal to 50%, the maximum possible value. But if the variation in the Generator was non-zero, you have no way of knowing what the actual maximum morph value will be for any individual particle. If you use the ‘**Morph at Max**’ mode you don’t need to worry about this since the value is stored in the particle and the question tests if that morph value has been reached.

The other three modes are simply three of the common modes with slightly altered names. You enter a value in the ‘**Morph Value**’ field and the particle’s current morph value is tested against that.

Particle Trail

This is a question which tests the length of the particle trail in different ways. The first three are effectively yes or no answers to the

following questions:

- Trail is at Max Length: the trail has reached its maximum length as specified in the Trail object.
- Trail Has Zero Length: the trail has no length at all, which might be the case if the **'Trails From Particle Birth'** switch in the Trail object was turned off.
- Trail is Less Than Max Length: the particle has a trail which has not yet reached its maximum length.

The other two question types are 'Trail Length (Percent)' and 'Trail Length (Absolute)'. The difference between them is that the 'percent' version tests the percentage of the maximum trail length that the trail currently is, while the 'absolute' version tests the actual length in scene units. Once you have selected one of these options you can then choose one of the common modes to test the length. The only addition is the **'Tolerance'** setting, which is only used when the mode is set to **'Equals'**. As explained at the start of this chapter, there is a problem with using **'Equals'** for floating-point numbers. The **'Tolerance'** setting is used when testing equality for the trail length to work around this problem by providing a little tolerance either side of the **'Length Value'** (percent or absolute) parameter.

See the example file [ch12_questions_particle_trail.c4d](#). This has particles with slightly different speeds to give some variation, then tests if the trail length ever equals 220 screen units. If that is the case, the particle turns red. If you play the scene, you see that no particle turns red, because the **'Tolerance'** is set to zero. Increase this to 1% and most turn red because the value now being tested is 200 +/- 1%, that is, a range of 198 to 202 units. To catch all of them you would need to set the tolerance to 1.5-2%.

12.2.5 Physical Data

All the questions in this section test the data which is set in the **'Extended Data'** tab, **'Physical Data'** quicktab in the emitter. The question uses the common question modes and tests the particle's physical data against the values you specify.

There is one parameter to note, which is that one item of physical data is labelled **'User Data'**. You can set this in the emitter and change the value (using the Physical Data modifier) to whatever you want, and test it in this question. The data item was added to avoid the technique of having to use an otherwise unused data item such as temperature when what was really wanted was a floating-point number with a user-determined value. Feel free to use this in whatever way suits your scene, it will not affect any simulation you have set up.

12.2.6 Other

This is a mixed bag of questions which don't fit anywhere else.

Boolean Input

X-Particles comes with a number of Xpresso nodes which you can use to change various aspects of the particles. Other Xpresso nodes can be used which don't use the XP nodes at all. If any node setup produces a Boolean output, it is useful to have a way of testing it in XP's question system. That is the purpose of this question. Unfortunately it appears to be broken in the latest version of X-Particles. The example file [ch12_questions_boolean.c4d](#) is supposed to change the particle colour when the speed exceeds 350 units. It is provided here so you can see how it is supposed to work; the Xpresso is working correctly but the value isn't being input to the Question object.

Current Time

This question tests the current scene time, so is not testing any particle data. The test modes are the common ones and you only need to input the scene time you want to test.

Particle Count

This tests the number of live particles in the scene. The common modes are used (for some reason the usual **'Is Greater Than'** has become **'Exceeds'**) and you enter the value to test against.

Python Script

With this question you enter a Python script which should return either True or False. The question will pass if it returns True. The example file [ch12_questions_python.c4d](#) shows this working. The script is a very simple one:


```
import c4d, xparticles
#welcome to the world of Python
```

```
def question():
    if particle.GetSpeed() > 350.0:
        return True
    else:
        return False
```


The scene has a Speed modifier which increments the speed. When it exceeds 350 units, the script will return True and the particle will change colour.

Random Probability

All this question does is generate a random number between 0 and 1 and tests if that number is less than the value in the **‘Probability’** parameter (internally the percentage is simply converted to a value between 0 and 1). The default value is 100%, which means that the question will always pass (actually, in theory if the random number generator produced a value of exactly 1.0 it would not pass, since that isn't less than 1.0...but the chance of that happening is minuscule). A value of zero means that it will never pass.

You can see that the smaller the **‘Probability’** value, the less chance that the question will pass. However, since the question will be re-tested each frame if it doesn't pass, eventually it is likely to pass if the scene is long enough.

You can use this question in one of two ways. First, you could use it as a simple random event which if passed will carry out some action. Perhaps the more useful way is to combine it with another question. For example, suppose you have a scene in which the particle speed is being tested. As soon as that test is met some action will occur. You can use the random probability question to ensure that the particles don't all pass at once, giving some variation to the scene. This is demonstrated in the example file [cb12_questions_randomprob.c4d](#), where you can see that even when the particle speed exceeds 350 units, not all the particles with that speed change colour immediately.

 The **‘Mode’** setting is only there for compatibility with older (much older) versions of X-Particles. For all new scenes it should be ignored and left at **‘Constant’**. This ensures that each time you play the scene it produces the same result. In the earliest versions of XP the result would vary each time the scene was played, and you can still see this if you change this setting to **‘Varying’**. Normally you would want the scene always to be same each time it was played. If you don't like the particular result you get, the way to change it is to alter the **‘Random Seed’** value in the Question object.

Number of Collisions

This question tests the number of collisions that have occurred between particles and scene objects. It does not include particle-particle collisions and nor does it include collisions using the Bullet system. Note that the number being tested is the total number of collisions for all particles, not for each individual particle.

12.3 Combining questions

Quite often you might want to combine questions so that an action takes place if two (or more) questions both pass or if at least one of two or more questions passes. This is done with **‘Sub-Questions’**. Sub-questions are in fact Question objects but with the **‘Actions’** and **‘Activate/Deactivate Modifiers’** sections removed from the interface. This is because the action which takes place applies to the combined questions and it is not possible to set a separate action for each sub-question in the combination.

Normally you would create a sub-question by creating a Question object then clicking the **‘Create Sub-Question’** button in the interface. This will create another Question object and add it to the scene as a child of the first Question object. You can have multiple sub-questions if required (but you can't have sub-questions of sub-questions). In the object manager it might look like Figure 12.10.

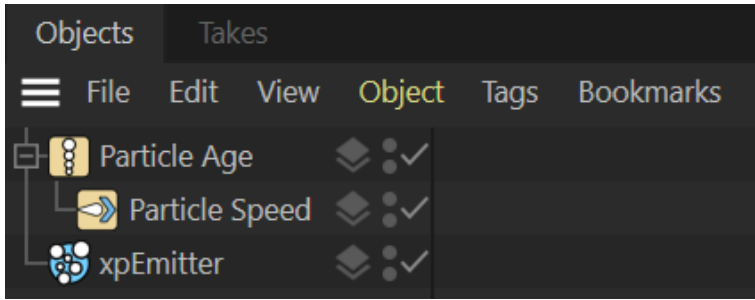


Figure 12.10. Arrangement of Question objects to give a sub-question

The questions and sub-questions all work as normal. The crucial thing though is the menu in the main Question object as shown in

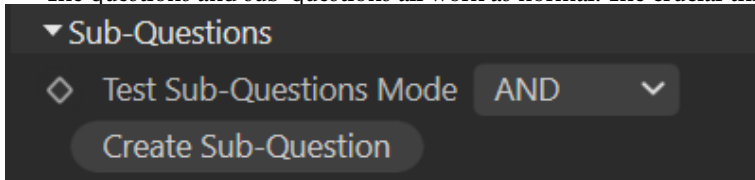


Figure 12.11.

Figure 12.11. Sub-question testing mode

This menu has two options:

AND: All the questions and sub-questions must pass at the same time to pass the combined question

OR: If any of the questions/sub-questions pass, the combined question will pass

The '**OR**' entry is of limited usefulness and it's usually '**AND**' that is used to ensure that two or more criteria are passed in order to pass the entire question and carry out the action.

12.4 Question order

The original way that questions worked in X-Particles was that if there was more than one question in the emitter's '**Questions**' list they would be executed in order but that if the second question passed before the first one did, the first question was deemed to have been bypassed and would not be tested - only questions lower down the list would be tested thereafter. It was also possible to force questions to be executed in strict order, so that the second question would not be tested until the first question had passed. In addition, once a question had passed it was not tested again, ensuring that questions higher in the list which had passed were not re-tested and potentially overwrite actions taken by questions lower in the list. However, it was possible to change that and force a question to be tested each frame regardless if it had previously been tested and passed.

Unfortunately in the latest version of XP that functionality seems to have been completely lost. All questions now seem to be tested each frame and the only thing you can rely on is that they will be tested in the order they occur in the list. This is certainly simpler to use but makes the XP question system less powerful. In any event, the '**Mode**' menu in the emitter Questions tab is now redundant and so is the icon which changes from yellow to blue when clicked. You can refer to the original XP reference manual to see how this was supposed to work if you are interested.

12.5 What happens when a question passes?

There isn't really much point in testing something unless some action will be taken as a result. Within each Question object there is a list labelled '**Actions**'. This is where you can add one or more actions to be taken if the question passes.

12.5.1 Actions

The Action object will be fully discussed in the next chapter. For now, it is sufficient to know that actions can enable or disable modifiers, directly alter particle data such as colour, control other objects, etc. To add an Action to a Question you can:

- click the **'Add Action'** button in the Question object, which will create a new Action object, add it to the scene, and drop it into the **'Actions'** list in the Question object
- add an Action object to the scene from the main X-Particles menu then drag it into the **'Actions'** list of the Question object

Several XP objects can also use actions themselves without needing a Question object but this is discussed more fully in Chapter 13.

When a question passes, it will call any actions in its **'Actions'** list as long as the action object has the yellow 'tick' icon. You can click that icon to turn it to a blue dash, in which case the action will be disabled until you click the icon again.

12.5.2 Activate/Deactivate Modifiers

The original purpose of the Action object was to activate or deactivate a modifier in action-controlled mode (see the 'Modifiers' section in Chapter 1, 'Basic Use' for more details). Many actions still do this - either that and nothing else, or with some more direct activity. To avoid having to create an Action object in the Question object simply to activate or deactivate a modifier, you can instead simply drag the modifier(s) you want to activate into the **'Modifiers to Activate'** list, or those you want to deactivate into the **'Modifiers to Deactivate'** list. This is simply for convenience and you can do exactly the same by using Action objects instead, if you prefer.

Summary

The X-Particles question and action system is a very powerful but easy to use system for controlling many aspects of XP. It's probably true that a node-based system would be even more powerful and flexible, but would also take a good deal of development and would not be as simple to use. The next chapter will look at the Action object, since without that the questions don't really achieve anything.

Chapter 13: Actions

In Chapter 12 we looked at the Question system in X-Particles and it was pointed out that questions aren't useful unless something happens when the question passes. To make something happen in such cases, we use an Action. This chapter shows how to invoke actions and how to use them for various purposes.

↓ The scene file(s) for this chapter can be found in the archive [examples_ch13.zip](#) on the book's website. Click here to [download the archive](#).

13.1 Basic principles

An action is something which takes place under certain circumstances. What actually happens depends on the nature of the action, but can be summarised as follows:

- the action changes the status of an action-controlled modifier
- or the action alters some of the particle's data such as speed, colour, etc.
- or the action affects some other XP object such as the emitter to change how it works

13.1.1 Changing the status of an action-controlled modifier

As briefly described in Chapter 1, 'How Modifiers Work', XP modifiers have two modes of operation. The default is **'Independent'**. In this mode, a modifier just works; provided that the particle is in its field of effect, the modifier will act on the particle. It is of course possible to prevent modifiers from working on particles in certain groups or even on all particles from a specified emitter, but in independent mode there is no control over individual particles. For that, 'Action-Controlled' mode is required.

In action-controlled mode, a modifier won't work on any particles unless it has first been enabled for a particle. It is possible to set criteria for when a modifier should start to work on a particle, and if those criteria are met, to turn on the modifier for that particle.

As an example, take a look at the example file [ch13_actions_acmod.c4d](#). This scene has a Gravity modifier in action-controlled mode. The emitter is set to emit particles with a speed between 100 and 200 units. The Question objects test if a particle has a speed of 175 units or more, and is 30 frames old. If the question passes, it triggers the Gravity modifier action. This action does two things: it sets the particle colour to red (which is entirely optional but makes it easier to see what is happening) and turns on the Gravity modifier for those particles which meet the two criteria.

What if instead of turning on a modifier so that it affected a particle we want to turn it off so that it doesn't affect the particle? We can do this by using the **'Effect on Particle'** menu. This has two entries. **'Modifier Will Affect Particle'** means that the action will turn on the modifier for a particle. The other entry is **'Modifier Will NOT Affect Particle'** which turns the modifier off for a particle. It's perfectly possible to turn a modifier on for a particle then turn it off again later on - and to repeat this as many times as you like. We'll look at that in more detail later.

i Note that if what you need to do is use a Question object to trigger an action to turn on or turn off a modifier, you don't actually need the Action object. In the Question object you can drag and drop the modifier into either the **'Modifiers to Activate'** list (to turn the modifier on) or the **'Modifiers to Deactivate'** list to turn it off. This has exactly the same effect as using an action to do this, except that there is no colour change since the action does that, and this feature is there purely for convenience. Action objects are preferred though if you want to reuse them elsewhere in the scene, because using these modifier lists only works with a Question object. See the section 'Triggering Actions' below for more discussion of reusing actions.

13.1.2 Changing a particle's data

Actions that change particle data don't need a modifier to make the change, they do so by acting on the particle directly. In the example scene [ch13_actions_acdirect.c4d](#) the setup is similar to the previous example. However, in this case there is no modifier in the scene to be turned on; when the question passes, the Action object alters the particle directly by setting the colour to green and the speed to zero.

13.1.3 Altering an object other than a modifier

A small number of actions can affect other XP objects such as the emitter, rather than working through a modifier. For example, see the file [ch13_actions_object_emitter_trigger.c4d](#). This uses a 'Change Emitter' action to force an emitter to emit a shot of particles.

How this works is discussed in more detail below.

13.1.4 Triggering Actions

In the earliest versions of XP it was envisaged that only Question objects would be able to trigger an action. However, it soon became clear that it would be useful if other objects could do so as well, and now there are a variety of objects which can trigger an action. In each case the principle of use is the same: one or more Action objects are dragged and dropped into a list of actions and the object will trigger those actions for any particle which meets the required criteria. For example, see the file [cb13_actions_network.c4d](#). This uses a Network modifier which is set to detect intersections between the trails following the particles. If an intersection is detected, an action is triggered which changes the particle colour. It's easier to see what is happening if you play this frame-by-frame.

Here is a full list of the objects which can trigger actions in one way or another (some of them have more than one condition which can trigger actions):

Object	Action trigger:
Modifiers	
xpAttractor	When the particle is closer to, or farther away from, a specified distance to the Attractor modifier.
xpAvoid	When a potential collision with an object to be avoided is detected.
xpBranch	When an intersection between branches is detected.
xpChangeGroup	When a particle's group is changed.
xpCover	In cover mode actions will be triggered as long as the particle is stuck to the object (this can be set to one frame only); in target mode, when the particle comes within a specified range of its target object.
xpExplode	When a particle is affected by the modifier causing the particles to 'explode'.
xpFollowPath	Two possible conditions, which can both be used at the same time. Actions can be triggered either when the particle first enters the pathway and/or when it comes to the end
xpFollowSpline	Two possible conditions, which can both be used at the same time. Actions can be triggered either when the particle is captured by the modifier and moved to the spline, and/or when it reaches the end of the spline.
xpFollowSurface	Two possible conditions, which can both be used at the same time. Actions can be triggered either when the particle is captured by the modifier and pulled to the surface to be followed, and/or when the particle leaves the surface.
xpHistory	Triggered when a stored history is played back and the playback is completed.
xpInfectio	Two possible conditions, which can both be used at the same time. One set of actions is triggered when a particle moves to the 'incubating' state and the other set when it moves to the 'infected' state.
xpMorph	When a morph is complete.
xpNetwork	On network trail intersection
xpRotator	When a particle reaches 'escape velocity' and escapes from the modifier.
xpSplineFlow	When a particle completes the flow path by passing through the last spline flow handle.
xpTrailsMod	If a trail is being shrunk the actions will be triggered when the trail reaches zero length.
xpTriggerAction	Actions will be triggered as long as the particle is within the field of effect of the modifier, subject to other conditions in the modifier's settings.
Other objects	
xpFlowField	When a particle leaves the flow field volume, as long as it is still within the field of effect of the modifier.
Tags	
xpCollider	When a particle collides with the scene object hosting the tag (note: does not apply to xpBullet collisions).

Table 13.1. Objects other than questions which can trigger actions

13.2 Types of Action

X-Particles divides its Actions into five different types:

13.2.1 Control Modifier Actions

There are 36 actions of this type, of which all but two simply turn a modifier on or off. The other two (**'Branching'** and **'Multilevel Spawn'**) have small additional functions in addition to activating or deactivating a modifier.

13.2.2 Direct Actions

This group has 17 actions. All but one (**'Control History'**) have two modes. They either work in **'Control Modifier'** mode, in which case they work exactly as the control modifier action type, or in **'Direct'** mode. This is more interesting because in that mode the action can change something in the particle directly, without the need for a modifier. As a simple example, the 'Change Speed' action when triggered acts directly on the particle to change its speed; no other object or modifier is needed. Remember that actions can be triggered in several ways, so you could use this action to alter a particle's speed following a collision with an object, for example.

13.2.3 Dynamics Actions

Currently there is only one action in this group, which affects the 'Sheeter' object. All it does is turn the Sheeter on or off in exactly the same way as if the Sheeter was a modifier (which internally it basically is).

13.2.4 Object Actions

Three actions which work on other objects - the Emitter, Generator and Trail objects - to change them in some way.

13.2.5 Other Actions

A group of miscellaneous actions which don't fit any other category! By far the commonest one used will be the 'Editor Display Only' action, which does exactly what it says - changes the particle colour and display shape. This is very useful when debugging a scene that isn't working correctly because you can use it to see when an action is triggered. All other Action objects can do this too, since they incorporate this action into their interface.

13.3 Action objects: common interface elements

All, or nearly, all, Action objects have certain elements of their interface in common. Figure 13.1 is an example using the Attractor Modifier Action. The settings are as follows:

13.3.1 Action Type

From this menu you select the general type of action you want - these are the types described in the 'Types of Action' setting above. As you can see from Figure 13.1, this action is a control modifier type.

13.3.2 Action list

In this example the action list is a drop-down menu which is labelled **'Control Modifier Actions'**. The menu itself is a list of all the actions of this type. If you change the **'Action Type'** then the menu label changes, so if the type is **'Direct Actions'** the menu is also labelled **'Direct Actions'**, and of course, the menu itself would change to be a list of all the actions of that type. You can then select whichever action is needed from the menu.

13.3.3 Add Modifier

This is a convenience feature. If you click this button, a modifier of the correct type will be added to the scene and in the Action object it will be dropped into the link field which holds the modifier to be affected by the action (see below).

All actions which affect, or can affect, a modifier will have this button. The **'Sheeter Object'** action also has it but in that case it reads **'Add Object'**, and in the **'Change Emitter'** action it reads **'Add Emitter'**.

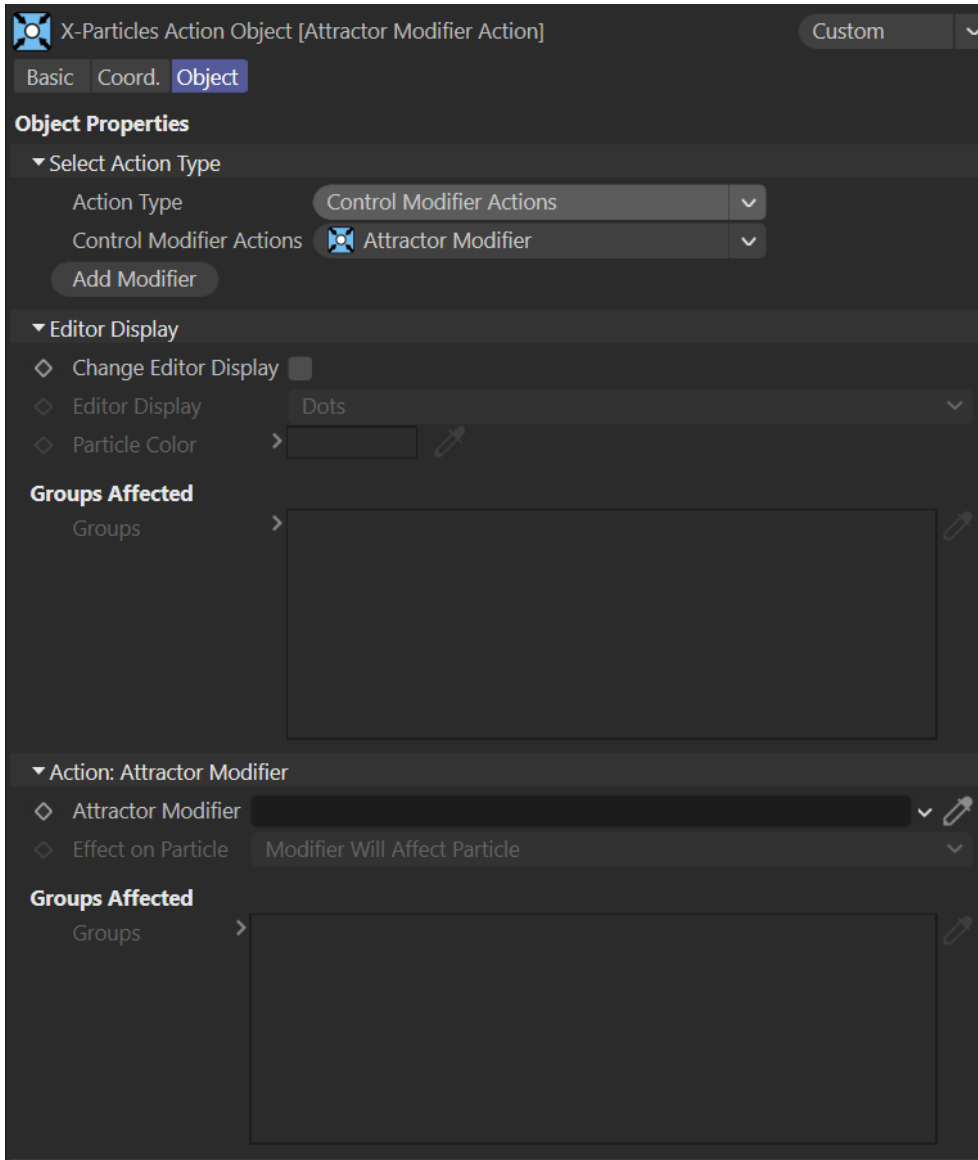


Figure 13.1. Action object common interface elements

13.3.4 Editor Display

This section is identical to the 'Editor Display Action' which is discussed below. All actions have this section.

13.3.5 Modifier link field

This is where you drag and drop the modifier to be affected. In this example it is labelled '**Attractor Modifier**' because that's the type of modifier being controlled; the name will change if you change the type of modifier to control. You can only add the correct type of modifier for the specified action into this field, so in this example you could not drag and drop anything other than an Attractor modifier.

All control modifier and direct actions have this field, as does the '**Sheeter Object**' action. For direct actions, the field is disabled until the '**Mode**' menu is changed to '**Control Modifier**' (instead of '**Direct Change**'). Some other actions may also have this field, such as '**Stop Following Spline**' and '**Unlink TP**'.

13.3.6 Effect on Particle

For control modifier actions, the direct actions in 'control modifier' mode, and the sheeter object action, this menu controls what the action will do. '**Modifier Will Affect Particle**' is the default and means that, when triggered, the action will turn on the modifier for the relevant particle(s). If it is changed to '**Modifier Will NOT Affect Particle**' the action will turn the modifier off.

13.3.7 Groups Affected

In this section you can control which particle groups are affected by the action.

You will see that there are two lists labelled **'Groups Affected'** in the action's interface. The first of these is in the **'Editor Display'** section of the interface, but in fact the list is unavailable (greyed out). This list is only available if the action is the **'Editor Display Only'** action. In all other cases, if you change the editor display, the groups that will be changed are those found in the second list.

If the action is a control modifier type, the modifier will only be turned on (or off) for the groups in the list; you can temporarily exclude a group by clicking the yellow tick icon so it becomes a blue dash.

As an example, see the file [ch13_actions_groups.c4d](#). This scene contains a Gravity modifier which is turned on by an action when a particle's age reaches one second (30 frames in this scene). There are two particle groups, one red and one blue. The **'Groups Affected'** list in the action contains only the blue group, so only those are affected by the modifier when it is turned on.

The **'Change Editor Display'** switch is also on so when the particle is one second old, its colour is changed to yellow. Since only the blue group is affected by the action, only the blue particles change colour.

i You could do the same thing in this file by dragging the blue group into the **'Groups Affected'** list in the modifier, instead of in the action. However, if you do that, although only the blue particles will be affected by the Gravity modifier, all the particles will turn yellow since the action is no longer restricted only to the blue group. It will in fact turn on the modifier for all particles, but the modifier will only work on the blue ones since the blue group would be in its own groups list and the red one would not.

Note that all control modifier, direct, and dynamic Action objects have this section. So do all the 'other' actions except **'Unlink TP'** where the effect is not appropriate for different groups to be affected differently. None of the 'object' actions have this section because they work on objects other than modifiers and don't affect particle data directly.

13.4 How-To: use actions to turn modifiers on and off as required

For this mini-tutorial we'll set up one of the most frequently requested procedures in X-Particles: making particles cover an object then move on to cover another one.

13.4.1 Basic scene

The basic scene is in the example file download as [ch13_actions_covertut_1.c4d](#). This just contains an emitter, a pyramid primitive and a sphere primitive. The aim is to use the Cover modifier to make particles cover the pyramid then move to cover the sphere.

13.4.2 Adding a Cover modifier

First, add a Cover modifier to the scene and rename it to 'Cover Pyramid'. We do this because another modifier will be added later and we want to be able to distinguish between them. In the Cover modifier, drag the pyramid object into the **'Target Object'** link field. If you play the scene, the particles move to cover the surface of the pyramid object. But how to move them to the sphere?

Now add a second Cover modifier and rename it 'Cover Sphere' then drag the sphere object into its target object link field. If you play the scene now, you get a peculiar (but not unattractive!) result where the particles move in a sort of loop, unable to decide which object to cover. This is because they are being acted on by both modifiers and neither one can override the other. This is not what is wanted; clearly, we need to be able to have just one modifier active at first then turn that off and make the other one active instead.

13.4.3 Switching to action-controlled mode

If you select the two modifiers in the object manager you can change them by using the **'Mode'** menu; by default it is set to **'Independent'** so change it to **'Action-Controlled'**. Make sure that both are changed then play the scene.

⚠ You'll see that now, nothing happens at all. This is due to a fundamental point with regard to action-controlled mode: a modifier in that mode will not affect any particles *until* it is turned on for each particle.

To do this, add a Question object to the emitter. By default this will be a particle age question, which is what we want. We need to turn the 'Cover Pyramid' modifier on as soon as the particle is created, so firstly change the **'Age'** setting to one frame and change the

'Mode' menu to **'Equals'**. This will ensure that the question passes once and once only, when the particle is first created. The question will look like this:

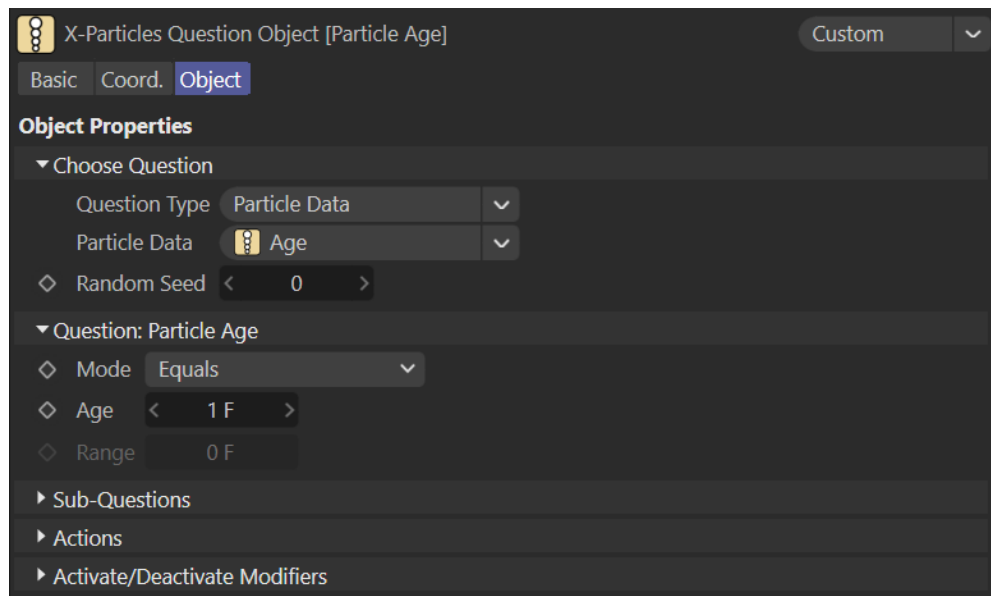


Figure 13.2. Question set to pass as soon as a particle is created

i Note that you can't use a test of the age being equal to zero frames. Due to the way XP works internally when a particle is created and its data is set to default values its age will be one frame, not zero.

13.4.4 Moving particles to the pyramid

To make passing the question do anything, we need to add an Action object. In the Question object, click the **'Add Action'** button, which will create a new Action object and will also add it to the **'Actions'** list. The action will be the default **'Editor Display'** type, but we want to control a Cover modifier, so select **'Control Modifier Actions'** from the **'Action Type'** menu, then choose **'Cover/Target Modifier'** from the **'Control Modifier Actions'** menu. Because we'll have several actions in the scene, rename it to **'Turn on Cover Pyramid'**. Finally, we need to tell the action which modifier to affect, so drag and drop the **'Cover Pyramid'** modifier into the **'Cover/Target Modifier'** link field in the action. The action will look like this when complete:

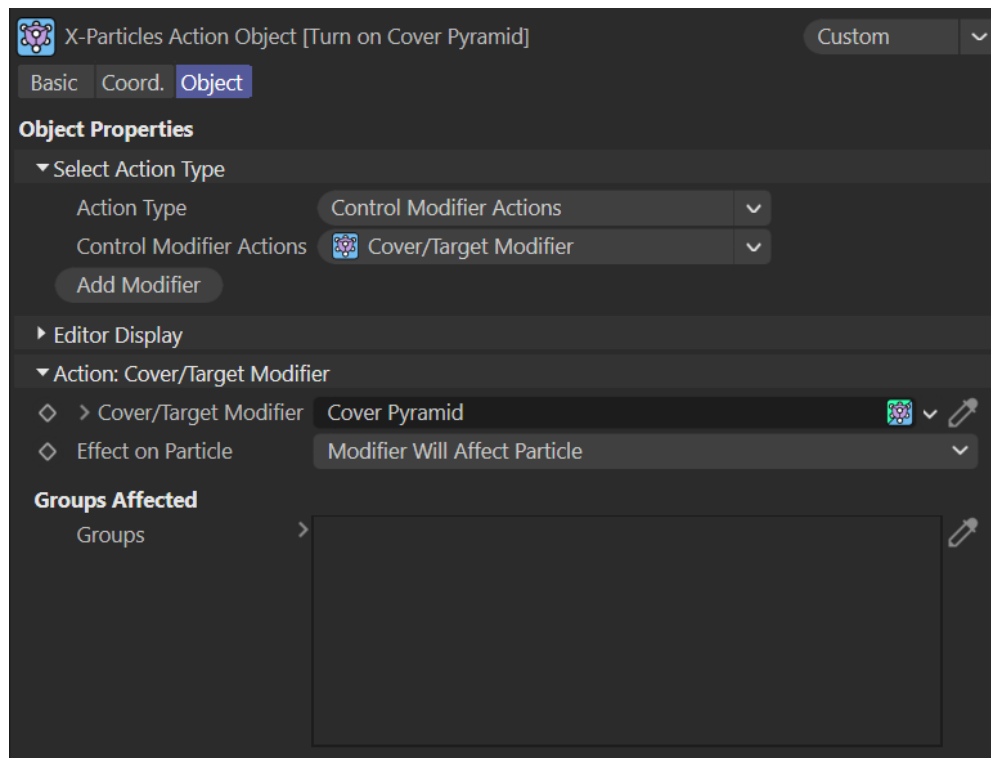


Figure 13.3. Cover Modifier Action ready to use

Play the scene. What you should see is a stream of particles moving to the pyramid and sticking to its surface, which is exactly what we had with the modifier in independent mode. The big difference though is that we can now release or ‘unstick’ the particles from the pyramid.

13.4.5 Releasing particles from the pyramid

At this point the particles are stuck to the pyramid and they’ll stay there even if the ‘Cover Pyramid’ modifier is disabled in the object manager, because internally a flag has been set which sticks them in place. This flag won’t be cleared if the modifier is simply disabled; to do that, we need to turn the modifier off for those particles which have been stuck. Doing this will clear the flag and release the particles to move elsewhere.

Select the ‘Cover Pyramid’ modifier (not the action) and switch to the **‘Actions’** quicktab. There is a list for actions which will be triggered when a particle is stuck to the target object. We can use that to release the particles, so click the **‘Add Action’** button to create a new action and automatically add it to the list. Make the action a Cover/Target Modifier action and rename it ‘Turn off Cover Pyramid’.

In the new action, drag the Cover Pyramid modifier into the link field as in section 13.4.4 above but make one further change: from the **‘Effect on Particle’** menu choose **‘Modifier Will NOT Affect Particle’**. You might also want to turn on the **‘Change Editor Display’** switch and choose some suitable shape and/or colour so you can be sure that the action is being executed. Then play the scene. It looks now as if the particles aren’t sticking to the pyramid at all, because they are being released as soon as they stick.

13.4.6 Moving particles to the sphere

Finally, we need one more Action object. Again select the ‘Cover Pyramid’ modifier, and add a second action to the **‘Actions’** list. Set it up again to control a Cover modifier, and rename it to ‘Turn on Cover Sphere’ but this time, drag and drop the ‘Cover Sphere’ modifier into the modifier link field in the action. The ‘Cover Pyramid’ modifier should now look like this:

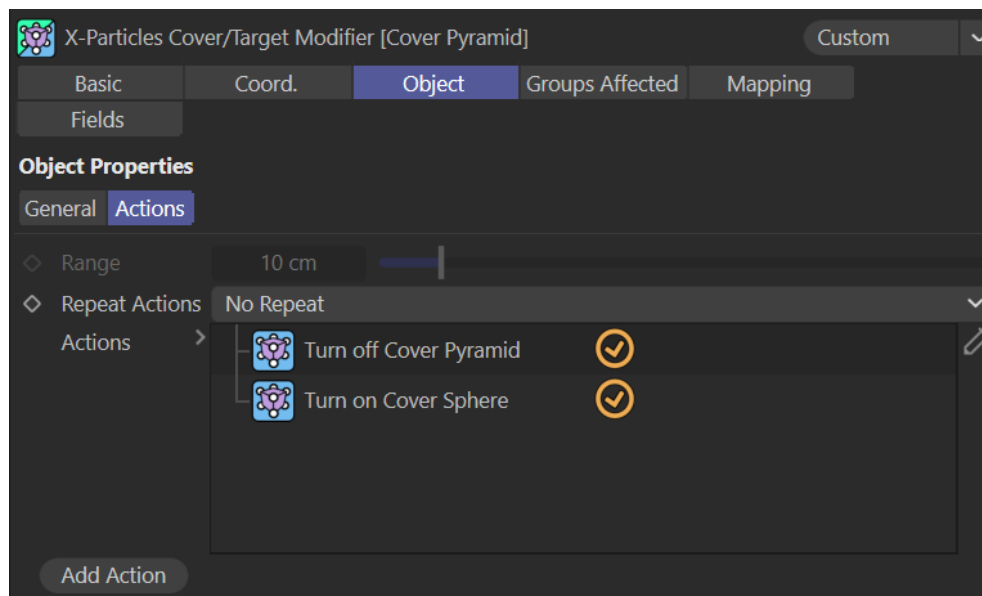



Figure 13.4. Cover Pyramid modifier with all required actions

When you play the scene, you should see the particles move to the pyramid then immediately move to the sphere where they stick because we haven’t turned off the ‘Cover Sphere’ modifier. The complete scene is in the download file as [ch13_actions_covertut_2.c4d](#).

There’s no limit on the number of objects you can use. Just remember that for each object you will need a modifier to move particles to that object plus two actions for each modifier, one to turn it on and one to turn it off (except perhaps for the final object where you won’t want to turn the modifier off if you need the particles to remain stuck to it). You can even make particles loop continuously round a series of objects. There’s an example file [ch13_actions_covermods.c4d](#) showing how this is done in the download archive. The final frame in that scene is shown in Figure 13.5.

 You can see the completed animation for the looping particles on the [book’s website](#).

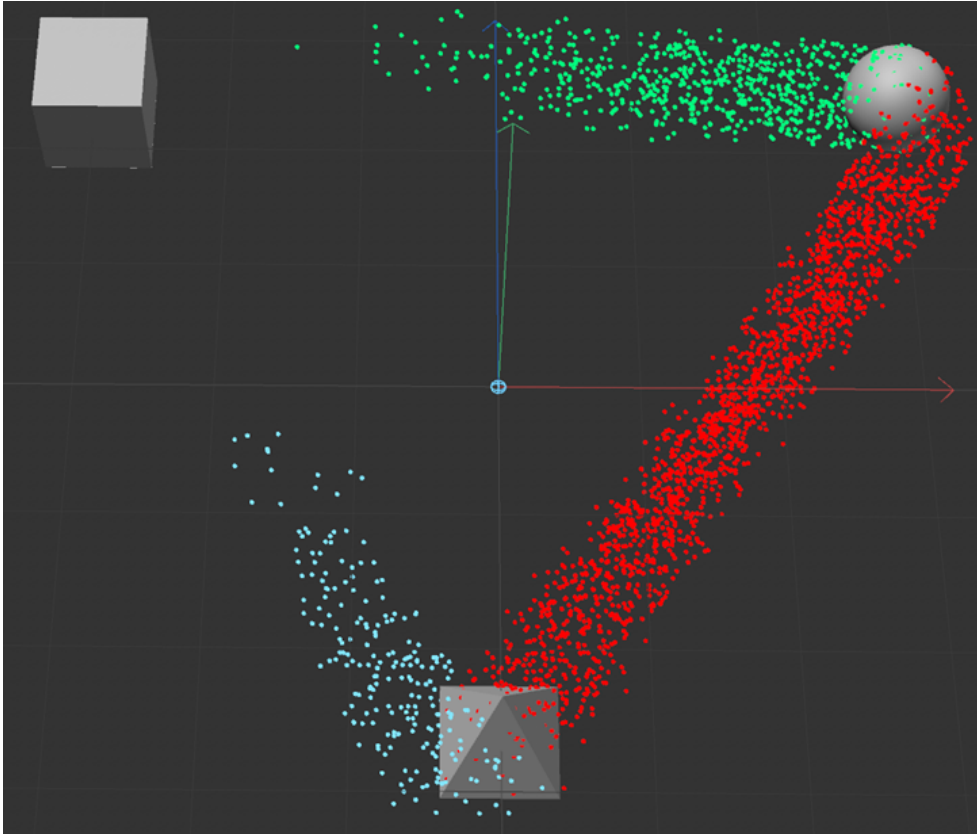


Figure 13.5. Using actions to 'loop' modifiers

13.5 'Other' Actions

It may seem slightly odd to start with the miscellaneous group of actions, but this group contains the default action you get whenever a new one is created, the **'Editor Display Only'** action.

13.5.1 Editor Display Only action

As its name suggests, all this action does is change the editor display of the particle, that is, its colour and shape. You can change either of both of these but you must turn on the **'Change Editor Display'** switch first.

Changing the editor display may have an effect on the animation. Changing the colour has an obvious effect if particles are to be rendered, while changing the shape may affect what happens if you use the Display Render object or you are using the XP implementation of Bullet dynamics, when the particle shape has a direct effect on collisions with other objects.

The interface of this action is also present in every other action in XP. This is mainly so you can see when an action has been triggered, in case things aren't working as expected and you need to check if and when an action is being triggered.

13.5.2 Output to Console action

This is potentially a useful action which can tell you exactly when an action was triggered, which particle triggered it, and which event triggered the action. This information is output to the console (that's the Cinema 4D console, not XP's own console).

See the example file [ch13_actions_output_to_console.c4d](#). The particles in this scene are set to follow an Arc spline and an action to output to the console is triggered when the particles reach the end of the spline. If you open the console (there's a button in the action interface to do that) and optionally clear anything already there (there's a button for that, too) then play the scene you can see the particles follow the spline and eventually reach the end and leave it. At that point the action is triggered. The result in the console looks as shown in Figure 13.6.

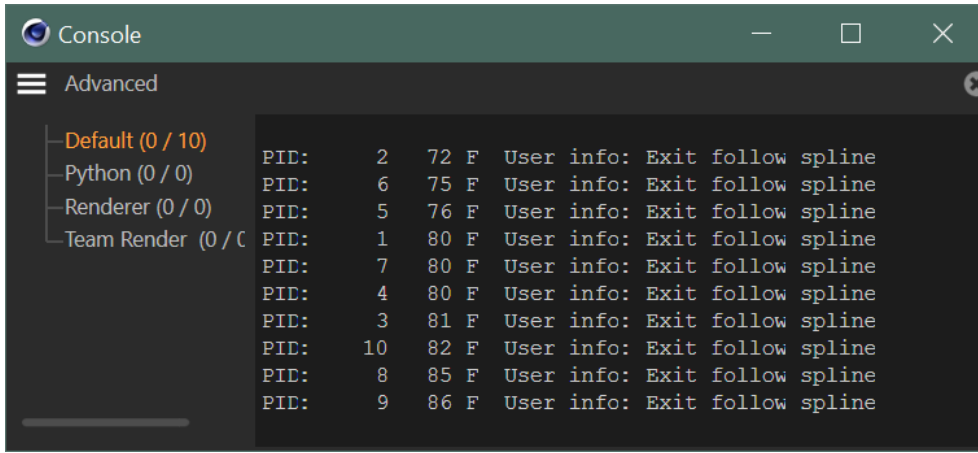


Figure 13.6. Result of the Output to Console action

The unique ID of each particle is displayed on screen and if you play the scene frame by frame you can see that the order in which the particles leave the spline is shown in the console. You could probably check this visually in this scene but now imagine doing that with a scene with thousands of particles...

There are a number of options for what is sent to the console including the unique ID of the particle (PID), the frame on which the action was triggered and the name of the action. There's also a text field for any additional information you might need but you have to enter that yourself.

13.5.3 Stop Following Spline action

The file in [ch13_actions_stop_following_spline.c4d](#) has the same setup as the example used in the output to console section. The action is used in conjunction with a Follow Spline modifier and when triggered it tells the particles to stop following the spline. A Question is used to turn on the modifier when a particle is created, then another question triggers the stop following spline action when the particle is 50 frames old. The action also sets the particle colour to green when triggered.

At first sight this is no different to turning off the modifier. If you change the action so that the modifier is simply turned off, you will see the same effect. However, the stop following spline action gives you a lot more control over what the particles do when they leave the spline. Simply turning off the modifier with an action causes the particles to continue with the speed and direction they had when the modifier was turned off. With the stop following spline action you can have several options for the particle direction and more for speed. They are all self-explanatory and a little experimentation with this file will show what the options do.

⚠ One point to be aware of is that in this file the output to console action is still present but you will note that the action is never triggered. This is because the particles don't leave the spline under control of the modifier but due to an external event, so the action is not triggered.

13.5.4 Unlink TP

XP can generate Thinking Particles (TP). Each TP has a corresponding X-Particles particle and follows its position, velocity, etc. At some point you might want to separate the two so that the TP no longer depends on an XP particle; that is what this action does. There is also an Unlink TP modifier, and this action can either unlink TPs directly or can turn on the modifier to do so.

13.5.5 Unstick from Source

Chapter 4 discusses emitting particles from an object and section 4.3.3 explains the use of the **'Stick Particle to Source Object'** switch. But what if you want to release the particle so it is no longer stuck to its source object? For that, you need this action.

The example file [ch13_actions_unstick_from_source.c4d](#) shows this working. The particles are stuck to the sphere object and there is a Trigger Action modifier (whose sole purpose is to trigger an action when a particle is within its field of effect) which moves along the world X-axis. When the falloff value reaches 100% at the position of a particle the particle is unstuck and moves with the velocity it was given when it was created.

Optionally you can force the particle to use the world speed and/or direction. If you use one of the world speed options you will have to turn on the **'World Speed'** switch in the emitter but even then you will find that the particles, though now unstuck, do not

move. This is because the sphere is not moving so the particles' world speed is always zero.

i You can use a '**Control Negate**' action to unstick particles instead. See section 13.8.1 below for discussion of that action.

13.6 Control Modifier and Dynamic actions

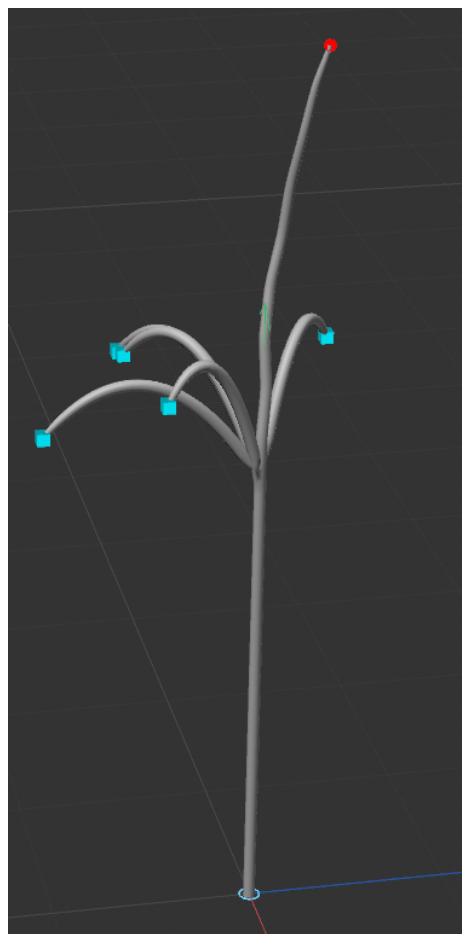
These actions all work by turning on (or off) a modifier for specific particles and their use was discussed in section 13.1.1 'Changing the status of an action-controlled modifier'. They all work in the same way and there is no point going through them one by one. However, two of these actions have additional functions which are discussed here.

The single Dynamics action, '**Sheeter Object**' works in exactly the same way as the control modifier actions - that is, it turns a Sheeter object on or off for specific particles. This works because internally the Sheeter object is a modifier to all intents and purposes.

13.6.1 Branching Modifier action

See the example file [ch13_actions_force_branch.c4d](#). This contains a Branching modifier in action-controlled mode with one sub-branch object, which is needed to generate some branches. If you play it, there is a single particle emitted and at first nothing happens. When the particle is 45 frames old, the Question object triggers a Branching Modifier action which turns on the modifier and changes the particle colour to red. You can see that the modifier is working because the 'stem' being generated will start to bend.

However, no branches are produced and this is because in the modifier's '**Branching**' quicktab the '**Branch In**' setting is set to 1000 frames - so no branches will be generated until that many frames have elapsed, which is way beyond the scene length. Using the action though, we can force a branch to be generated. In the action interface, turn on the '**Force Branch**' switch and you will see that as soon as the modifier is activated a single branch is generated. You can get multiple simultaneous branches by changing the '**Number of Branches**' setting in the branching quicktab of the modifier. Doing that and making some additional changes to use particle groups, a Gravity modifier and a SplineMesher object might show something like Figure 13.7. The scene producing that image is in the download archive as [ch13_actions_force_branch_multi.c4d](#).



Using this action you can force a new branch at any time. In the file [ch13_actions_force_branch_on_collide.c4d](#), there are two branching modifier actions. The first turns on a branching modifier from particle birth, but as before there are no branches being generated. When the particle collides with the plane object, the second action is triggered which forces the generation of two branches.

13.6.2 MultiLevel Spawn Modifier action

The use of this action will probably be a lot clearer once we discuss the corresponding modifier, but for the sake of completeness the action will be explained here.

Essentially, a MultiLevel Spawn modifier spawns (generates) particles from a source particle. If more 'levels' are added to the modifier, spawns can be generated from the initial spawns, then more particles spawned from those spawns, and so on. The matching action has two modes of operation. The first is the conventional turning the modifier on or off, but the second will alter settings within the modifier itself. (Not in the particles' data but actually in the modifier.)

See the example file [ch13_actions_mlspawn.c4d](#). When the source particle is created a Question object has two actions to trigger, but in this file one of them is initially disabled. There is also a second Question object with one action, also initially disabled. The single active action simply turns on the modifier and 15 frames later (a time set in the modifier) a single burst of 50 spawned particles is generated.

Now enable the action '**Disable Level 1 in MLS Modifier**'. On playing the scene you can see that the action has been triggered because the particle turns red, but no spawns are produced. This is because level 1 is the first and only level in the modifier so if it is disabled

Figure 13.7. Example of using the Branching modifier action to force branching to occur

no spawns are generated.

If required we can turn this level back on again. The second Question object tests for a particle age of 45 frames and will trigger the action **'Enable Level 1 in MLS Modifier'**. If you turn that action on in the object manager, so that all three actions are now enabled, you can see that the level 1 spawns are not generated at frame 15 because the level is disabled. However, at particle age 45 frames the level is enabled again and immediately spawns a burst of particles.

The modifier itself can have multiple levels of spawning. You can specify which level to control and you also have the option, when disabling or enabling a level, to affect just that one level or that level plus all the levels under it.

Finally, note that once you have disabled or enabled a level that level is now completely under the control of the actions. If you want to stop that and let the modifier take over control of that level again, use the third mode of operation called **'Release'** which puts the level back under the modifier's control.

13.7 Object actions

There are three of these actions, so called because they work on XP objects other than modifiers. For the Change Generator and Change Trails actions you must drag and drop the object to be affected into the link field in the action interface.

13.7.1 Change Emitter action

This action affects an emitter, but if you look at the interface you see that there is no link field for an emitter to affect. (There is a link field for a trigger emitter, but that is something different.) So which emitter is affected?

Emitter Settings

For the first two settings - **'Emit New Particles'** and **'Visibility'** the affected emitter is either the one which contains the Question object which triggers the action, or if the action is triggered in some other way, the emitter which emitted the particle which triggered the action is the one affected.

An example might make this clearer. In the file [cb13_actions_emitter_noemit.c4d](#), there are two emitters but there is no Question object present. In the action, the switch **'Emit New Particles'** is turned off, so when the action is triggered, the emitter will stop emitting any more particles. What actually triggers the action is collision with the cube object so as soon as a collision between the cube and the blue particles occurs, the blue emitter emits no more particles. However, the emitter emitting the red particles is unaffected, because none of the particles it creates collide with the cube and therefore the action is not triggered for the red emitter:

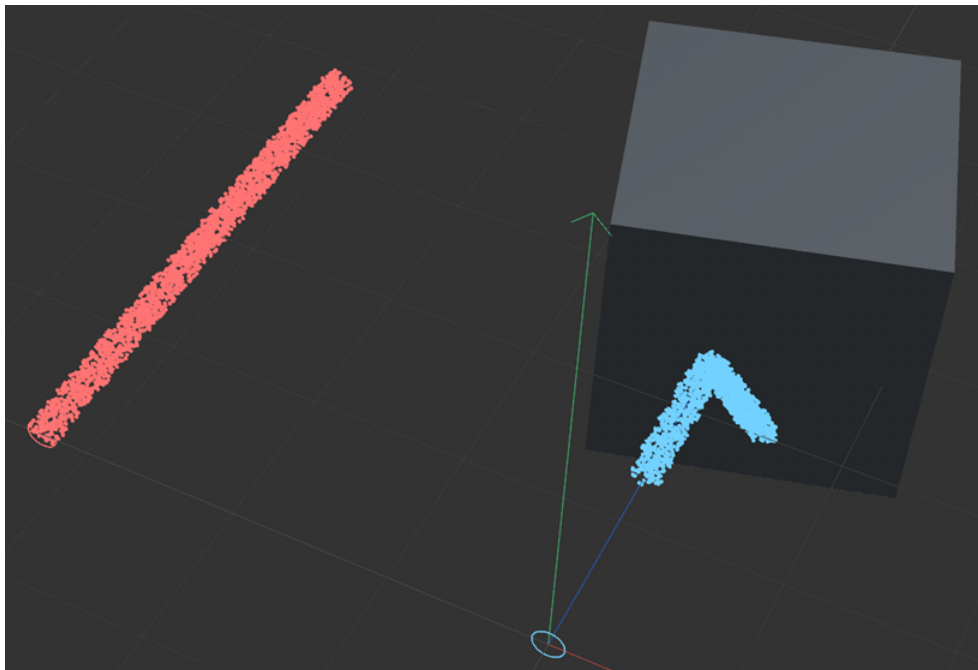


Figure 13.8. Change Emitter action triggered on collision with an object

If you move and resize the cube so the both red and blue particles collide with it, you can see that both emitters will stop emitting particles when a collision occurs without needing a second Action object to do this.

Turning on the **'Emit New Particles'** switch if it has previously been turned off will make the emitter emit more particles again. You can do the same with the **'Visibility'** menu - if this is set to **'Invisible'** all existing particles become invisible, but they are still present in the scene.

Trigger Emitter

The file [cb13_actions_object_emitter_trigger.c4d](#) shows what this does. There are two emitters in the scene but only the blue one (the 'Source Emitter') emits any particles at first. At frame 30 in the animation a Question object triggers an action which forces the red emitter to emit a single shot of 100 particles. How does this work?

First, we must provide an emitter to be triggered, which is the red emitter in this scene, and is named the 'Triggered Emitter'. For this process to work, that emitter must have two important settings: in the Emission tab, **'Emission'** must be set to **'Trigger'** and **'Trigger Count'** must be set to **'Set by Action'**. If either of these conditions is not met, the triggering function will not work.

In the change emitter action, the 'Triggered Emitter' is dragged into the **'Emitter'** field. This specifies to the action the emitter we want to control. The **'Mode'** in this case is set to single shot with a shot **'Count'** of 100. The Question object in the 'Source Emitter' triggers the action when the scene frame equals 30 frames. At that point the triggered emitter is forced to emit a shot of 100 particles.

Instead of triggering a single shot of particles, you can set the action **'Mode'** to **'Hold Trigger'**. This will cause the triggered emitter to emit 100 particles each frame from then on. This won't stop unless you use another change emitter action with the mode set to **'Release Trigger'**

13.7.2 Change Generator action

If you look at the example file [cb13_actions_object_generator.c4d](#), you see that there is a Generator with two child objects, a cube and a sphere. The Generator is set to generate only the first object, that is, the cube.

A Question object triggers a Change Generator action when the scene is 45 frames old. It simply changes the index of the child object to generate to 2, which will cause the Generator to generate spheres instead.

Very importantly, note two things. Firstly, the existing cubes are not replaced by spheres, so individual particles are not affected. (To do that you would use a Geometry modifier or Change Geometry action.) Second, the action has not altered the settings in the interface of the Generator object - it has changed internal data to ensure the correct object is generated, so that even though the interface still indicates that object 1 should be generated, in fact object 2 is produced.

The other settings the action can change are whether to animate an object, so you could turn off animation at the start in Generator itself then turn it on as appropriate. Do remember that this will not affect the animation of individual particles, it is an all-or-none effect.

You can also force a change in the current morph target and specify the target index to use. See Chapter 9, section 9.5 ('Object Morphing') for details of how to morph generated objects.

13.7.3 Change Trails action

In this action there are two types of function: you can alter the generation of trails on particles which exist when the action is triggered, or you can alter the way particles are connected by the Trail object.

Trail generation on existing particles

The example file [cb13_actions_object_trail.c4d](#) is an example of the first type of function. The **'Shrink Trails'** switch is turned on, so when the action is triggered the trails of any existing particles will shrink to zero, at a rate of one trail point per frame. There's no control over the rate of shrinkage or a minimum length at which shrinking will stop. You can opt to kill the particle when the trail length reaches zero if desired.

Note that in this scene the action is triggered just once, when the scene time is 30 frames. At that point trails for all existing particles (those generated by the linked Trail object of course - other Trail objects would not be affected) will start to shrink, but the trails of particles created after the action has been triggered will never shrink.

Another option is to stop the Trail object generating any trails on existing particles by changing the **'Trails'** menu to **'Don't Generate Trail'**. You can turn them on again later with another action with the **'Trails'** menu set to **'Generate Trail'**.

i Trail shrinking only works if there are no connections between particles (i.e. the trail algorithm is set to **'No Connections'**).

Connection algorithm

In addition to, or instead of, altering trail generation you can also change the connection algorithm. If you turn on the **'Change Trail Algorithm'** switch you can select the connection type from the **'Algorithm'** menu. This menu contains most of the connection algorithms found in the Trail object, and selecting any algorithm except **'No Connections'** gives you some (but not all) of the parameters for the algorithm found in the Trail object.

The example file [cb13_actions_object_trail_algo.c4d](#) shows this working. At frame 45, the connection algorithm is changed from **'No Connections'** to **'Nearest by Index'**. Unlike the trail generation settings, this change affects all particles, both existing and new ones (because you can't have a mixture of connection algorithms with different algorithms for different particles). As with the change generator action, this change in algorithm does not change the interface of the Trail object.

The final setting is an interesting one. Suppose you have particles connected using the **'Nearest by Distance'** algorithm. If particles change distance from one another, for example if a Turbulence modifier is being used, the closest particles to others may change, causing the connections to change; some existing ones may break and new ones may be created. You can stop this from happening by using the **'Connections'** menu set to **'Lock Connections'**. Once you do that, the existing connections are maintained even if they would normally be broken due to particles moving away from one another. In addition no new connections will be made for any particle. You can undo this effect with another action, this time with **'Connections'** set to **'Unlock Connections'**.

The example file [cb13_actions_object_trail_lock_connections.c4d](#) demonstrates this. If you watch it closely, you can see that connections are made and broken again as the particles move apart. After frame 90 however, when the action is triggered, no new connections are made but existing ones remain even with the distance between particles would normally break them. At frame 150, the connections are unlocked, existing connections are broken if the particles are too far apart, and new ones are made.

! Note that this action only works on three trail algorithms - all points to all points, nearest by index and nearest by distance. It has no effect on the other algorithms.

13.8 Direct actions

Of the 17 actions in this group, all but one work by default in direct mode without requiring a modifier to do anything, but can also work like any control modifier where they turn the relevant modifier on or off. The exception is the **'Control History'** action which will be discussed separately. The main point of these actions however is to act directly on particles to change the particle's internal data, such as radius or speed.

Most of these actions are straightforward and only require a brief discussion.

13.8.1 'Simple' direct actions

All these actions work very simply to change some sort of internal data. They include:

Change Custom Data

If you have set custom data up in the emitter (see Chapter 5, section 5.5 'Custom Data') you can change the value of the data with this action. Simply identify the data item to change with its name and/or ID value, specify what type of data it is, and in the **'Operation'** menu choose to either set a new value or to increment or decrement the existing value. In either case, the new value to set or to be added to the existing one is found in the **'Value'** field. Negative values should be used if you want to subtract from the current value of the data.

Change Geometry

This action will only work if the emitter is linked to a Generator object. You can choose to:

- change the index of the Generator's child object to be generated for a particle, where the first child has index 1, the second index 2, and so on.

- randomly select from the child objects of the Generator; you need to specify the number of child objects in the **'Number of Objects'** field and you can set this to lower than the actual number of child objects if you want to use only a subset of them. However, if you set that value to greater than the number of child objects, not all particles will have an object since in some cases the number selected will be higher than the number of available objects.
- select child objects sequentially starting with the first one. Once the total number of child objects is reached, no more objects will be generated; to start again from the first child object, select **'Sequential with Wrap'** instead of **'Sequential'** mode.

Change Group

This action changes the particle's group to the group which is dragged into the **'New Group'** link field. You can select any of the new group's parameters to be set in the particle. For example, if the particle is red but the new group is coloured green, and the **'Color'** switch is turned on, the particle colour will change to green when the action is triggered. To copy over the new group settings, simply turn on the switches for all the parameters you want to use from the new group.

Change Infectio

Only used if the particle is affected by an Infectio modifier. With that modifier, a particle is always in one of three states: not infected, incubating, or infected. You can use this action to change the state of the particle to whichever one is required.

Change Life

This action will alter the particle life in some way. The most frequent use is to set **'Operation'** to **'Kill Particles'**. When triggered, the action will immediately kill the affected particles.

You can instead increase (or decrease) the particle lifespan. The lifespan will be increased (or decreased) by a randomly-selected value between the **'Min Change'** and **'Max Change'** settings. Note that if the new lifespan is reduced so that it is below a particle's age, the particle will immediately die.

Alternatively, you can set a specific lifespan to the vale in the **'New Lifespan'** field. Finally, you can change the lifespan relative to the particle's age. For example, suppose the particle's lifespan is 60 frames and its current age when the action is triggered is 30 frames. If the operation is set to increase the lifespan, and the minimum and maximum values are both 10 frames, the new lifespan will be 70 frames ($60 + 10 = 70$). But if the operation is to set lifespan relative to age, the new lifespan will be 40 frames (current age of 30 plus $10 = 40$).

Change Lights

The lights referred to here are the lights generated by a Sprite object. With the action you can change the illumination from the light, or the visible brightness, or both. See the file [ch13_actions_direct_lights.c4d](#). A Question object triggers this action when the particle age is 45 frames. If you play the scene you can see the lights abruptly dim when they reach that age. In this file you can also try changing the visibility with or without changing the illumination.

Change Scale

This action does a little more than just change the scale. It can in fact change any or all of three parameters, radius, mass and scale. To change these, turn on the relevant switch then enter a new value plus any required variation.

i As an aside, particle mass is included in this action since although mass is found in the emitter as part of the physical data, the Physical modifier action is not a direct action so there is no other way to change the mass directly.

Change Speed

This will alter the particle's speed which either changed relative to the current speed (so if the current speed is 100 units and the **'Speed'** setting is also 100 units, the new speed will be 200 units) or an absolute value which simply sets the particle speed to the value in the **'Speed'** setting. You can add variation to the new speed in both cases.

As an optional extra you can change the particle direction as well. Turn on the **'Change Direction'** switch then either opt for a randomly-chosen direction or one of the six orthographic directions. If you want to change direction without altering the speed, set the speed mode to relative and the **'Speed'** and **'Variation'** values to zero.

Change Spin

! To use this action, you must first have enabled particle rotation in the emitter. See Chapter 5, section 5.1 'Particle Rotation'.

This action can do two things. If '**Operation**' is set to '**Rotate**' the action will rotate the particle once, using the other settings in the action. If it is set to '**Spin**' it will rotate the particle each frame.

The '**Rotate**' operation immediately rotates the particle to the value in the '**Spin/Rotate Amount**' setting. See the file [ch13_actions_direct_spin.c4d](#). In this file the amount to rotate is 45 degrees on the heading (H axis) and at particle age 30 frames, when the Question object triggers the action, the particle rotation immediately snaps to 45 degrees on H. You can either make this an absolute setting, so it would always be set to 45 degrees on H no matter what the existing rotation was, or relative, in which case the setting value is added to the existing rotation.

In that example file, if you change the operation to spin, then the particle starts to spin on the H axis when the particle is 30 frames old. In this case, the particle will spin by the 'Spin/Rotate Amount' setting in one second. You can see in this file that the spin starts at age 30 frames and rotates by 1.5 degrees each frame ($45/30 = 1.5$) and will do this forever unless stopped by using another action with the spin value set to zero.

Change Sprites

This action will only work if the emitter is linked to a Sprite object. The first option ('**Generate Sprite**') will turn off sprite generation for the affected particle if that switch is turned off.

Secondly you can change the type of sprite generated. Note that if you do so, the parameters of the sprite will be those found in the Sprite object. You can see this working in the file [ch13_actions_direct_sprites.c4d](#). In this file, the Sprite object is set to generate cubes, but when the action is triggered at particle age 30 frames, the sprite type is changed to a text sprite, using parameters for the text which are pre-set in the Sprite object.

In addition you can change the material applied to the sprites. To do this there must be a list of materials in the Sprite object's '**Texture List**'. In the action you would then turn on the switch '**Change Material**' and set the index of the material to use. This is also shown in the example file, where the first texture (white) is applied to the cube sprite but the text sprite has the second texture (gold) which is changed by the action. A screenshot is shown in Figure 13.9.

Control Morphing

Morphing is discussed in detail in Chapter 9, section 9.5 'Object Morphing'. The use of the Morph modifier is also discussed there, but there is also a '**Control Morphing**' direct action.

The first two settings are identical to those found in the Morph modifier. '**Change Target Index**' does the same as the '**Change Morph Target**' switch in the modifier (see section 9.5.3, 'The Morph modifier'). If the switch is turned on, the Generator will use as the morph target the object in its '**Morph Targets**' list with the index specified in the '**Morph Target Index**' setting, which is identically named in both the morph modifier and action.

'**Set Morph Amount**' is off by default but if it is turned on morphing will take place up to the value in the '**Morph Amount**' field (the same as the '**Morph Max**' field in the Generator object). In other words, if you are morphing a cube to a pyramid and this value is set to 50%, the morphing will only progress half way before stopping.

Control Negate

The Negate modifier and action work identically. The only difference is that the modifier works on all particles in its field of effect while the action works when triggered by a Question object or some other event.

Both do two things: first, they let you clear some of the particle's internal flags. These flags are internal switches which tell an object such as the emitter or a modifier to manage the particle in certain ways. For example, there are several flags which instruct the emitter to freeze particle movement, spin or scale, others to stick the particle to objects, and so on. Some of these can be cleared by the Negate modifier and action; the 'freeze' flags can be cleared, and so can three flags which cause the particle to stick to an object under various conditions.

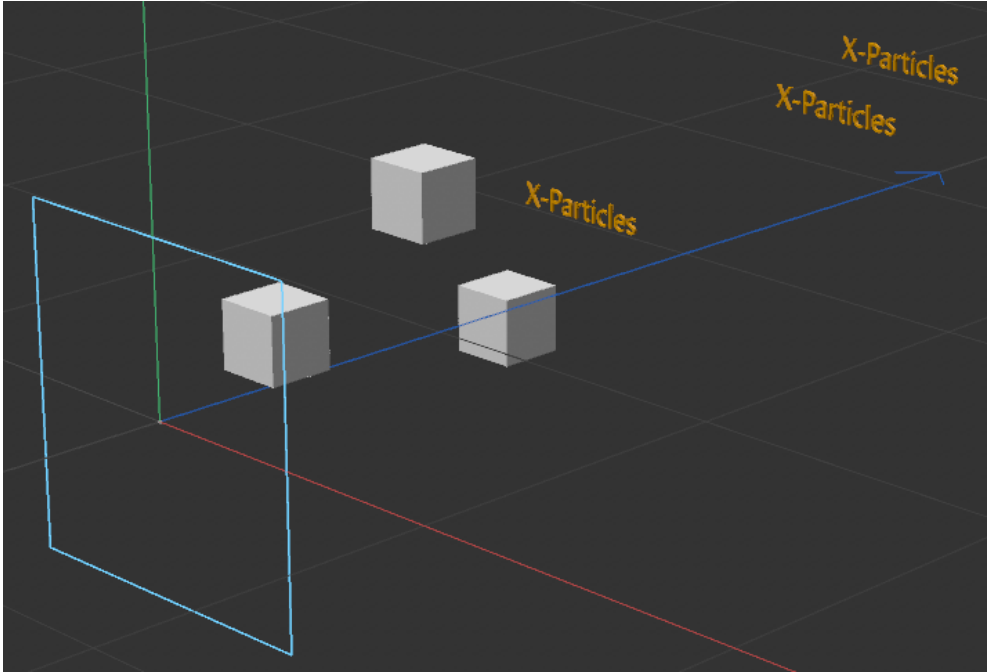


Figure 13.9. Changing the sprite type and material

The other thing the Negate action and modifier can do is break constraints between particles or with an object. This will be more useful when we discuss constraints in detail.

13.8.2 Actions with other functions than altering particle data

Control Spawning

This action works in a very similar way to the Spawn modifier, and the various parameters will be discussed in detail, with examples, in a later section dealing with particle spawning. There are one or two differences between the modifier and action however, and it is worth listing these here.

Feature	Difference between action and modifier
Allow Self-Spawning	Allowed in the action but not recommended for the modifier. This is discussed in detail in the section on spawning later in the book.
Spawned particle offset	An offset can be added to the spawned particle position if using the modifier. The action only has a random position offset switch which in any case doesn't appear to do anything.
Spawn number timing	The number of spawned particles can be set to per second or per frame in the modifier, but not in the action, where the number of spawns is generated each time the action is triggered.
Spawned particle position	An additional setting in the action can set the spawned particle position to Bullet dynamics collision points (using the XP implementation of Bullet, not the inbuilt Cinema 4D one).
Min and Max intervals between spawns	The modifier defaults to spawn particles each frame but can be set to allow longer time intervals between spawns. This is not relevant to the action which will only generate spawns each time it is triggered.
Inherit Color	An additional option in the action causes the spawned particles to inherit their colour from the source particle which triggered the action.

Table 13.2. Differences in spawning features between the Spawn modifier and Control Spawning action

Explode Particles

This action and the corresponding modifier will 'explode' a mass of particles in random directions from a specified point. The action is much simpler and only lets you set the speed of the exploding particles and, if necessary, to unstick them from an object they might be stuck to.

Freeze Particles

This action will freeze particles by setting internal flags to freeze any combination of movement, spin or change in scale. It can also

set the speed to zero, since freezing a particle doesn't set the internal speed to zero, it merely prevents the emitter from moving the particle, in which case unfreezing the particle will let it move again with whatever speed it had before it was frozen.

The action does have one control the modifier does not: **'No Trail Gap'**. The problem is that when freezing a particle the order in which events take place in X-Particles is that if the particle has a trail the trail is stopped from extending in length, then the particle is moved, and then it is frozen. This may cause a gap to appear between the tip of the trail and the particle. Turning on this switch will prevent that from happening.

Note that frozen particles can be released by using this action with **'Action of Particles'** set to **'Unfreeze'**, or as explained above, by using a Negate modifier or action.

13.8.3 Control History

The Control History action is unlike all the other direct actions in that it doesn't have the usual modes of either direct action or controlling a modifier. Instead, it always requires a History modifier and works by controlling that.

⚠ When you select this action from the **'Direct Actions'** menu in the Action object, its name in the object manager will be **'History Modifier Action'**, not **'Control History Action'**.

The History modifier itself stores particle data as the scene plays then at the relevant time replays that data back into the particle - so the particle data eventually reverts back to what it was when the modifier started to affect it. As an example of what it can do, see the example file [ch13_historymod_indep.c4d](#). Here, the modifier is working in independent mode (no control history action used). There is a Turbulence modifier to vary particle velocity and a Color modifier which changes the particle colour from blue to red. As the scene plays, the data is stored and then at frame 60 (scene time, not particle age) the data is played back. You can see the particles retrace the path they took and the colour reverts back to the original blue. When playback is complete the particles behave as normal again and the Color and Turbulence modifiers also work again as expected.

This is fine, but you have no control over when the playback starts other than setting the scene time to start playback in the History modifier. Using the Control History action lets you specify exactly when playback should start for individual particles.

See the example file [ch13_actions_direct_history.c4d](#). The History modifier is set to action-controlled mode but although it hasn't been turned on by an action, as would be the case for all other action-controlled modifiers, it still records the particle data. Playback starts when the particle collides with the cube object. The Collider tag triggers a Control History action which plays back all the recorded data. Once playback is complete the particle starts to move normally until it collides again with the cube, and this can be repeated indefinitely until the particle fails to collide with the object.

⚠ Important: there appears to be a small bug in the history system which can prevent it from working correctly when using an action. To see this, in the example file select the history modifier action and change **'Playback Using'** to **'Time'**, then play the scene. The playback time in the action is set to 10 frames, so you would expect the playback to be very short but it remains at 100%. This is happening because in the modifier **'Playback Using'** is set to **'Percent'** but in the action the equivalent setting is now set to **'Time'**. The same problem will arise if the action uses percent but the modifier is set to time. To get round this problem, ensure that **'Playback Using'** has the same setting in both the modifier and the action. And yes, in the modifier in order to change **'Playback Using'** you will have to change the modifier back to independent mode, change **'Playback Using'** as required, then put the modifier into action-controlled mode again.

Summary

That concludes the examination of Action objects in X-Particles. You can see the very considerable power that actions give you to change particles, turn modifiers on and off, and affect other objects. The most important point though is that using actions can be extremely granular - you can even affect specific, individual particles if you need to.

Chapter 14: The System object

The System object is a way of organising X-Particles objects in your scene. It was originally envisaged as just a glorified Null object, but it's quite a bit more than that. You can use it to make groups of objects easier to control, isolate them from one another, and as a convenient method to add more objects to the scene.

The scene file(s) for this chapter can be found in the archive [examples_ch14.zip](#) on the book's website. Click here to [download the archive](#).

14.1 Creating a System object

All you need to do is click on the object in the XP menu or tool palette and it will be added to the scene. In addition, it will create a number of other objects as seen in the object manager in Figure 14.1.

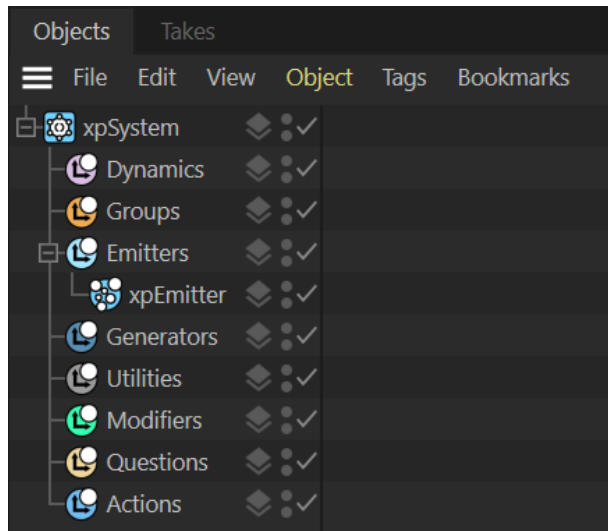


Figure 14.1. Objects added when the System object is created

In addition to the System object itself, a new emitter is created along with eight special null objects.

14.1.1 Folders

When first created, the System object creates eight special null objects - null in that they don't have any on-screen presence and don't do much other than create other XP objects. They are intended to act as folders, so that all your modifiers are in the same place, all the generators are in another place, and so on. All these folders, with the exception of the Questions folder, have menus with which to create the objects they are intended to contain. So for example, in the Generators folder, you have a menu which lets you create any of XP's generators as shown in Figure 14.2.

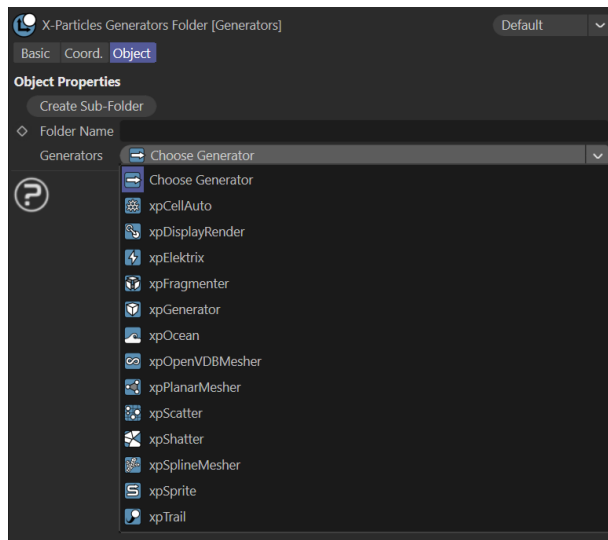


Figure 14.2. Adding an object from the Generators folder menu

You can use this menu to create the required Generator object and add it to the scene; the new object will be placed in the correct position in the System object hierarchy. The Modifiers and Actions folders actually have several menus which divide up the modifiers and actions into their various types. The Questions folder cannot create any questions though, because they can only be created by an emitter. However, if you use an emitter in the System object hierarchy to create a question, the Question object will be added to the scene and placed as a child object of the Questions folder.

There is one extra convenience feature: if you disable the folder in the object manager this will automatically disable all the child objects within the folder. Re-enabling the folder will enable all the child objects again (even objects that were already disabled when the folder object was disabled).

! If you use the 'Create Cloth' command from the Dynamics folder or the System object itself, you must provide the object to use as the cloth object by dropping it into the 'Cloth Object' link field before using the command, or nothing will happen.

Note that you can delete any of these folders and they will be recreated if necessary. For example, if you delete the Modifiers folder, and then want to add a modifier you can use the System object to do this. The System object will look for the Modifiers folder within its hierarchy and if it doesn't find it, will recreate it then add the new required modifier to it.

Finally, you can still create objects from the main XP menu and drag and drop them anywhere into the System object hierarchy if you wish - you don't have to create the objects using the System object if you don't want to.

14.1.2 Sub-folders

Within each folder object there is a button labelled 'Create Sub-Folder'. This will create a child object of the folder and by default this is named 'Sub-Folder'. This is to allow even further organisation of your XP objects. If you have a lot of actions, for example, you might want to divide them into groups depending on what they do. You can rename the sub-folder to something more specific but the sub-folder feature is purely for convenience; it has no other functionality. As with folders, disabling the sub-folder in the object manager will disable all its child objects as well.

14.2 System object functions

14.2.1 Disabling XP objects

In some cases when developing a scene you need to turn off some or all XP objects in the scene. If you aren't using a System object you have to disable each object manually, but if they are in the hierarchy of a System object you can simply disable the System object in the object manager and all active XP objects in its hierarchy will be disabled automatically. Any objects which are already disabled won't be affected. If you then re-enable the System object all the disabled objects which were previously enabled will be enabled again; objects which were already disabled when you disabled the System object will remain disabled.

By 'active' objects is meant the objects in the dynamics, emitters, generators, utilities and modifiers groups. Particle group, question and action objects are not disabled because on their own (with no emitters, all of which will have been disabled) they have no effect.

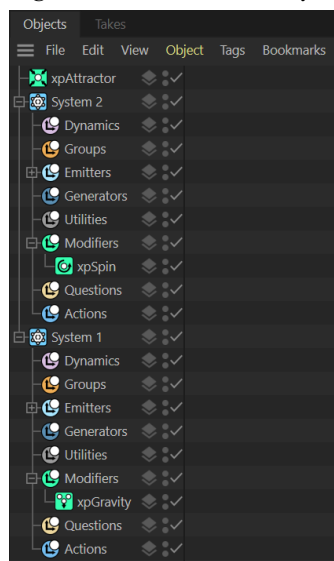
You can turn this feature off by turning off the 'Auto-Disable XP Objects' switch but there's no real advantage in doing so because this functionality is so useful. It makes it easy to have more than one System object with varying objects and/or different settings in the objects, and switch between them by disabling or enabling the System objects and their child objects with one mouse click.

14.2.2 Screen appearance

The System object has no points or polygons but it can draw a non-rendering icon in the scene if desired; turn on the **'Icon in Viewport'** switch to display this. The icon is not drawn if the System object is disabled, so if you have more than one System object you can use the screen icon to remind you which ones are active (it helps if you change the icon colours, of course). You can alter the colour and size of the icon from the System object's interface.

14.2.3 Restricting modifiers and/or deformers

If you have a System object containing modifiers, and other modifiers elsewhere, either outside any System object or within the hierarchy of another System, they will all still work on every emitter within every system object. So for example, in the arrangement in Figure 14.3 there are two Systems each with an emitter and a modifier, plus a third modifier which is outside both Systems:



All three modifiers will affect the emitters in both System objects, but this might not be what you want. You might prefer that the emitter in 'System 1' is only affected by the modifier(s) in that System and no others. To do that, you would turn on the **'Only Modifiers in Same System'** switch in 'System 1'. Now only the Gravity modifier which is in 'System 1', and neither of the others, would affect the emitter in 'System 1' but the emitter in 'System 2' would still be affected by all three modifiers in the scene. You can do the same with deformers by using the **'Only Deformers in Same System'** switch.

This scene is in the download archive as file [cb14_system_modsinsamesystem.c4d](#).

Figure 14.3. Multiple System object and modifiers

14.2.4 Adding XP objects

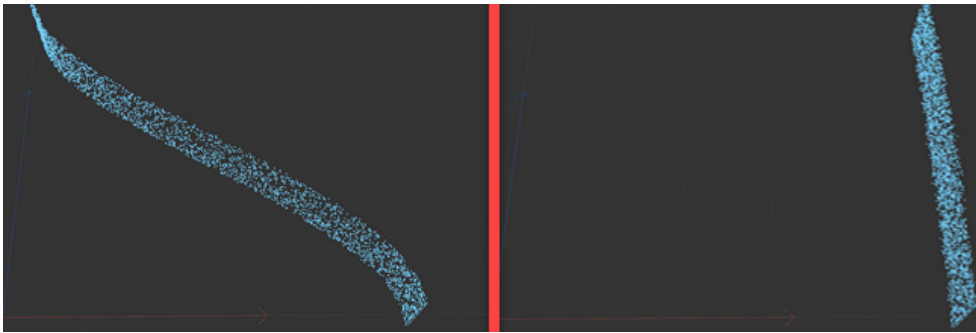
You can add most other XP objects except particle groups and questions using the menus in the System object interface or in the various folders (except questions). Any new objects created this way will be added to the hierarchy under the respective null object. Add questions using the emitter; the question will be placed in the correct folder in the hierarchy.

14.2.5 Global Transform

This tab was added to overcome a problem when moving or rotating an emitter. Suppose you have an emitter which is emitting particles and you want to keyframe the position of the emitter. What will happen to the position of the particles? Clearly, any new particles emitted after the emitter has moved will be emitted at the new position of the emitter - but what about particles which already exist? In the example file [ch14_system_anim.c4d](#) the emitter (which is not animated) is in the hierarchy of a System object which is animated to move along the X axis. You can see in the final frame of the animation shown in the left half of Figure 14.4, that new particles are emitted at the new emitter position, but the existing ones are not moved.

This happens because each particle has its own position in the 3D world which is a global position and is not related to the position of the emitter after the particle has been created.

In some cases this will be exactly what you want to happen, but in others you might want all the particles to follow the movement of the emitter. This is what the global transform function does. In the example file, look at the **'Emitters'** list in the global transform tab of the System object. The emitter is already there but has a blue dash against it, so is not affected by the global transform. Click to the blue dash to change it to a yellow tick and play the animation. The same frame as above now looks like the right half of Figure



14.4.

Figure 14.4. Global transform of particle positions. Left side, transforms are off; right side, on

You can move and rotate the System object and any emitters in the list will be transformed (i.e. moved and/or rotated) and any existing particles will also be moved so that their position relative to the emitter remains the same. The System object does this by converting the global particle position into one relative to the emitter position.

⚠ Note that for this to work, you must animate the System object, not the emitter itself. Animating the emitter will have no effect.

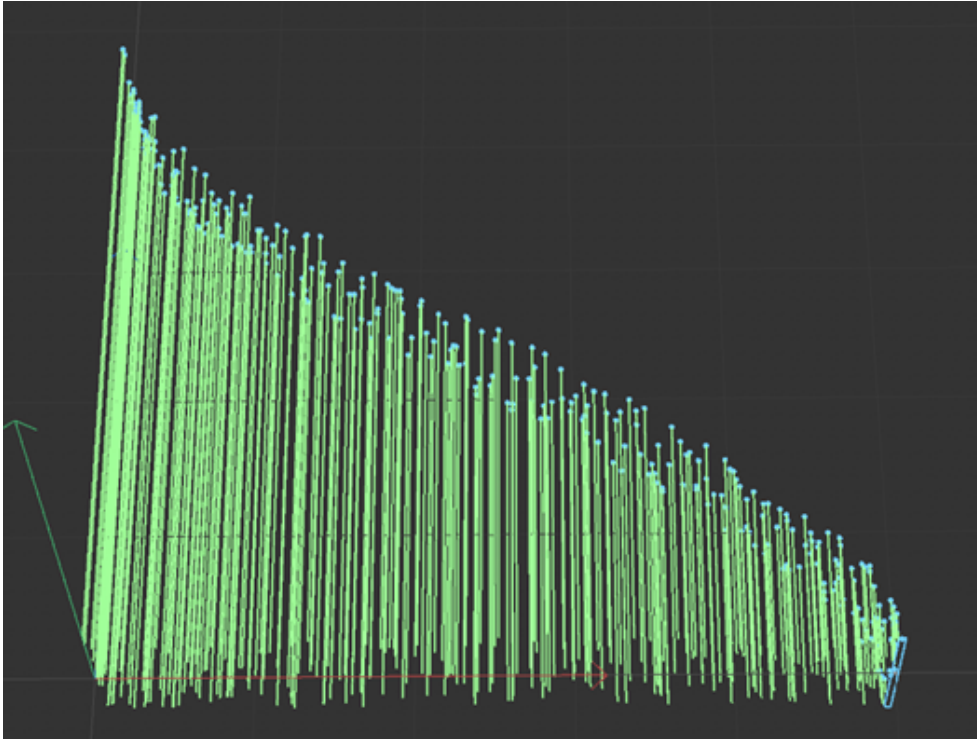
One final point concerns what happens to generated objects associated with the particles. For the Generator and Sprite objects this is no problem since the generated object is always located at the particle position, whatever that is. But for the Trail object, there is an added issue. The points which make up the trail spline have a position relative to the position of the Trail object, not to the emitter and not to the particles. If the emitter is not being transformed and the Trail object is outside the System hierarchy, you would see the result in Figure 14.5 (the scene setup is the same as in the above example file). Here, you can see that the trails are unaffected by the movement of the system object. This is what you would expect; the already-emitted particles are not being moved because the emitter is not being transformed and neither are the points of the trail spline as they are relative to the Trail object which is not moved along with the System object.

If the emitter *is* transformed but the Trail object remains outside the System object hierarchy, the result is shown in Figure 14.6. This happens because the trail points are still unaffected by the System object movement but existing particles are now being moved.

Switching it around, if the Trail object is inside the System hierarchy and the emitter is *not* transformed the effect is shown in Figure 14.7. Now existing particles are not being moved but the points of the trail spline are, because the Trail object is a child of the System object and each point position is therefore relative to that of the Trail object which in turn is relative to the System object.

Finally, if the Trail object is inside the System hierarchy and the emitter is being transformed, we see the result in Figure 14.8. You can see that any of these combinations might give you a desirable effect but for the trails to be kept in the same position relative to their particles even when the System object is animated you need to ensure both that the emitter is being transformed and the Trail object is somewhere in the hierarchy of the same System object.

If you want to try this out, there's a basic file in the download archive, [cb14_system_anim_trail.c4d](#) to experiment with. Try rotating



the System object instead of (or as well as) moving it and see what happens.

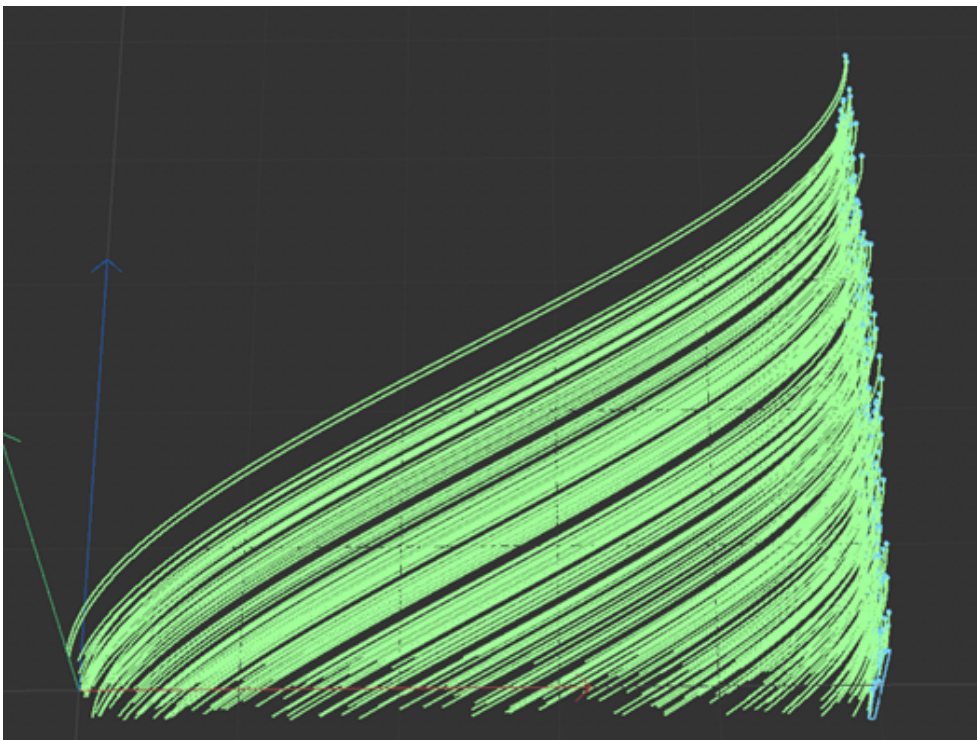


Figure 14.5. Emitter is **not** transformed; Trail object **outside** System object

Figure 14.6. Emitter **is** transformed; Trail object **outside** System object

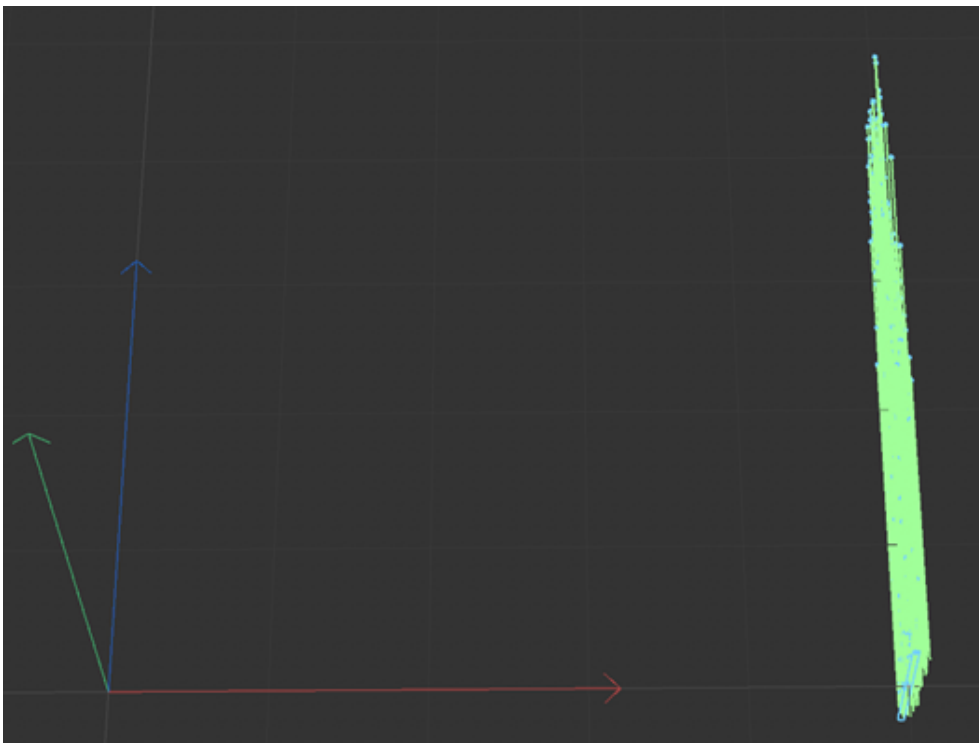
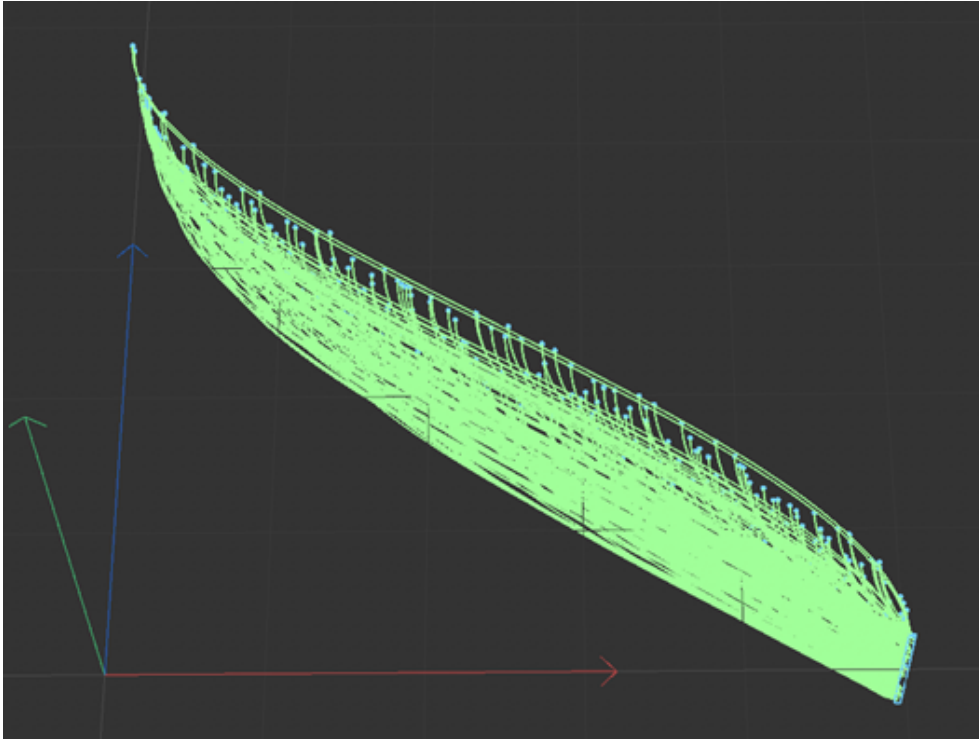


Figure 14.7. Emitter is **not** transformed; Trail object **inside** System object

Figure 14.8. Emitter **is** transformed; Trail object **inside** System object

Chapter 15: Modifiers Part 1

Modifiers are a core feature of X-Particles. They act to modify particles in some way. This can be anything from simply changing particle data such as speed or radius, to simulations in their own right, or the ability to generate more particles from the source particles affected by the modifier.

In this and the subsequent chapters on modifiers I won't be working through every setting in each modifier. That's properly the job of a reference manual, which this book is not. Instead, I will concentrate on three things:

- what the modifier does in broad terms
- why you might want to do that
- and how to achieve the desired result

These chapters will therefore be on the lines of a collection of 'how-tos' on the basis that if you work through each section you will have a better understanding of what the modifier does rather than simply reading a list of all its parameters.

The scene file(s) for this chapter can be found in the archive [examples_ch15.zip](#) on the book's website. Click here to [download the archive](#).

15.1 Basic principles

Almost all modifiers have the following features in common.

15.1.1 Mode

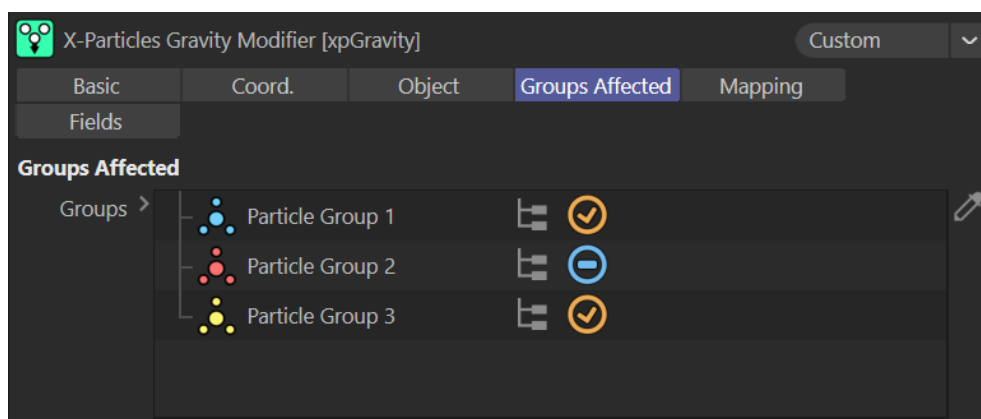
All modifiers have a '**Mode**' menu which has two settings, '**Independent**' or '**Action-Controlled**'. These have already been discussed in Chapter 1, section 1.2.2 'How modifiers work' and Chapter 13, section 13.1.1, 'Changing the status of an action-controlled modifier'. For the difference between these modes see those sections for a full discussion.

15.1.2 Groups Affected

This is separate tab in the modifier interface which contains a list of particle groups. Depending on the contents of this list you can restrict the effect of the modifier to specific particle groups. The tab is available for all modifiers except the UnlinkTP modifier.

For example, in this scene there are three particle groups and a Gravity modifier. The Gravity modifier '**Groups Affected**' list looks like Figure 15.1.

Figure 15.1. Groups affected list with one group not affected by the modifier (blue dash icon)



The two groups with yellow tick icons will be affected by the modifier but the one with a blue dash icon will not. The result is shown in Figure 15.2.

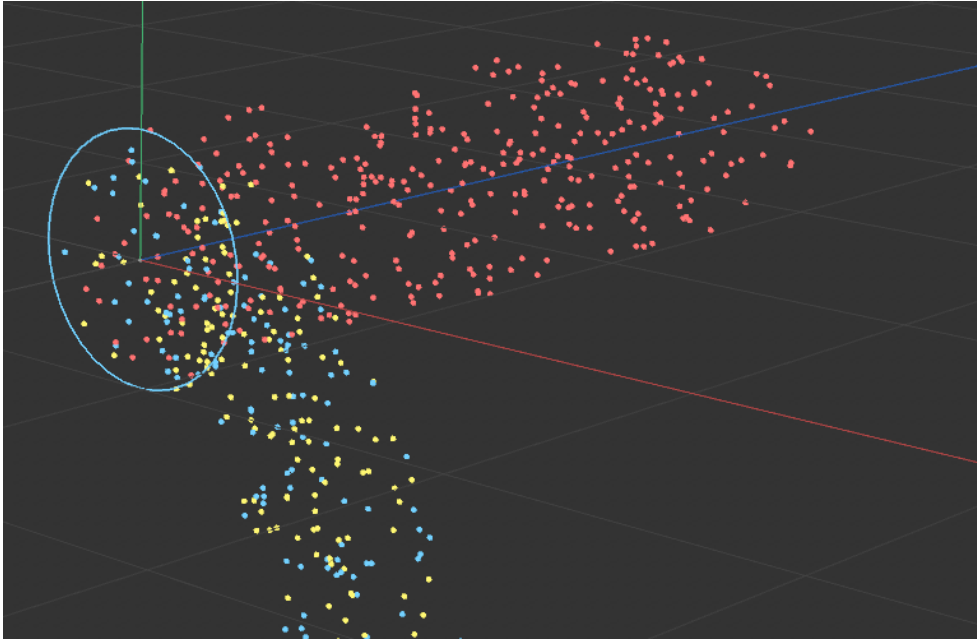


Figure 15.2. Gravity modifier affecting two groups out of three

The blue and yellow particles are in the groups affected by the Gravity modifier and therefore ‘fall’ under its influence but the the red ones are not affected, and the red particles do not fall.

This behaviour is as you would expect, but consider the same scene where only the red particles are in the list l(Figure 15.3).

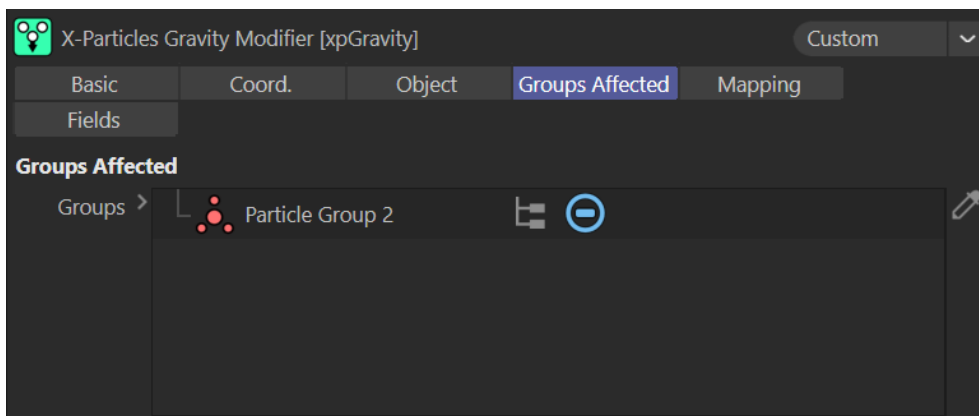


Figure 15.3. List with one group which is not affected by the modifier

The behaviour of the particles is seen in Figure 15.4.

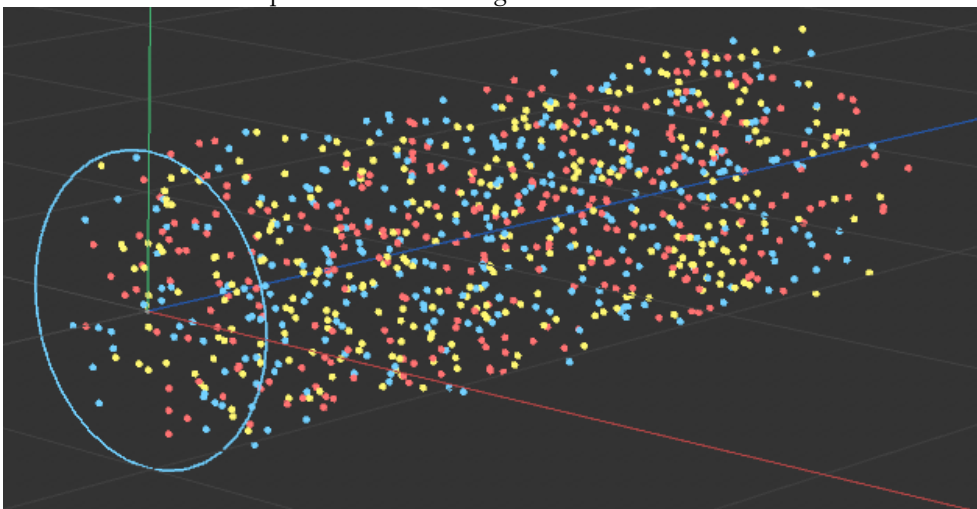


Figure 15.4. Result of the groups list shown in Figure 15.3

No particles seem to be affected by the modifier. This is counter-intuitive. You would expect the red particles to be unaffected because of the blue dash, but what about the other groups? They aren't in the list at all, so should they be affected by the modifier or not?

This is easier to understand if you know what X-Particles does with the list internally. The algorithm goes like this:

- if there are no modifiers in the list, i.e. the list is empty, all particle groups are affected by the modifier
- if there are any groups at all in the list, only those will be affected by the modifier
- but if any of the groups in the list have a blue dash icon, they are unaffected

In the above scene therefore, we can see that the list is not empty (so only the group with red particles will be affected by the modifier) but in fact the red group has a blue dash so it is unaffected. The result is that no groups are affected by the modifier, which is what you see in Figure 15.4.

That leaves a question: if you have multiple groups and you want to exclude just one from a modifier's effects, how do you go about that? You can't add that one group to the list - if it has a yellow tick icon, it will be the only affected group but if it has a blue dash icon none of the groups will be affected. The only solution is to add all the groups to the list and give them all yellow ticks apart from the one to exclude, which gets a blue dash.

15.1.3 Mapping

The Mapping tab is present in most, but not all, modifiers. The ones that don't have it are:

- History
- Kill
- Python
- Trails Modifier
- Unlink TP
- Weight
- MultiLevel Spawn

The tab gives access to a powerful feature called data mapping. What this does is let you set the value of certain modifier parameters based on some other value. For example, you could set the particle speed to vary depending on its radius. To do this you would use a Speed modifier with the speed mapped to the radius. The example file [ch15_mods_dmap.c4d](#) shows this working. If you play the scene you can see that the larger particles move faster than the smaller ones. The final frame is seen in Figure 15.5.

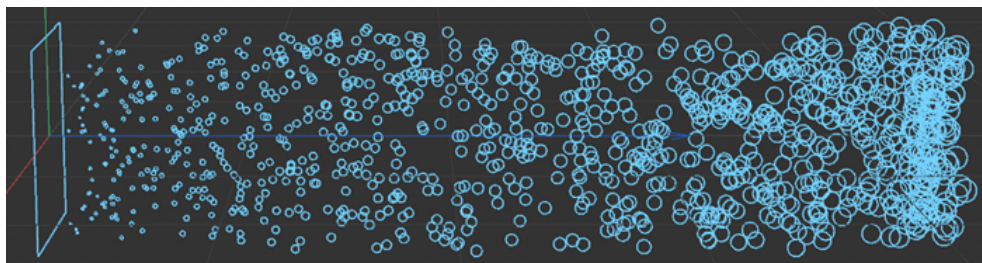


Figure 15.5. Mapping particle speed to radius

Because data mapping is a large topic in its own right, we'll leave further discussion until its own chapter later in the book.

15.1.4 Falloff/Fields

In Cinema 4D pre-R20 this tab is labelled Falloff, but subsequently it is named Fields. The job it does is the same in both cases, it's only the implementation which is different. Since most users are probably now using R20 or later, I will only refer to fields here, but it all applies to falloffs in R19 and earlier as well.

All the modifiers have this tab except Kill. By default, the field of activity of any modifier is infinite - the modifier will work with equal strength on any particle anywhere in the scene. You can restrict this using a field of whatever kind fulfils the requirements of the scene. So the first thing a field (or falloff) can do is set an area in the 3D world where the modifier will affect particles but outside of which it will not.

On its own this is a fairly crude feature but in many modifiers the field can do more than that. All fields/falloffs return a value to

the object which uses them. When provided with a position in the 3D world, the value returned is zero if the position is outside the field of effect. If the position is within the inner zone of the field where there is no falloff, the returned value is 1.0. If it is inside the field of effect, but not in the inner zone, the returned value will be greater than zero but less than 1.0.

Figure 15.6 shows how this would work for a Box field. Anywhere outside the box, the returned value is zero. Anywhere within the inner box, it is 1.0; in between the outer box and the inner box it will be somewhere between those values, the exact value returned depending on how close the position is to the inner box.




Figure 15.6. Falloff values returned from a Box field (or Falloff, pre-R20)

What does the modifier do with the returned value? If it is zero, the particle must be outside the field of effect and the modifier will not affect the particle at all. For any other value, in many cases (but not all by any means) a parameter used by the modifier will be multiplied with the falloff value.

An example will show this happening. In the example scene [ch15_mods_falloff.c4d](#) there is a Speed modifier with a Box falloff (this scene was built in C4D R19, but it will work equally well in R20 and higher versions). The modifier itself sets the speed of the particle to 150 units/second in the '**Speed Value**' setting. If you play the scene, you will see that when a particle enters the box, it slows down abruptly, but doesn't stop completely. As it moves further into the box, it starts to speed up, until when it is inside the inner box it will be moving at the top speed of 150 units/second. Then, when it leaves the inner box and heads towards the outer box it will start to slow until when it reaches the edge of the outer box it comes to a halt. This happens due to the following sequence of events:

1. The particles start with a speed of 150 units/second, set by the emitter.
2. Outside the box the falloff value is zero and the modifier does not affect the particles at all.
3. When the particle arrives inside the outer box the returned falloff value is very small (0.001 would be a typical value); this is then multiplied with the speed to be set by the modifier (150 units/second) giving a very low speed, but just above zero.
4. As the particle moves further into the box the falloff value increases, and so too does the speed.
5. Eventually the particles enter the inner box, the falloff value is then 1.0 and the particle speed is set to 150 units/second.
6. As the particles leave the inner box the falloff value decreases and therefore the speed which is set decreases until at the edge of the outer box it reaches zero.

You might wonder why the particles which arrive at the edge of the box (step 3 above) don't immediately have their speed set to zero. This is actually an effect of subframe emission which causes the particles to have initial starting positions which are slightly different from one another. This means that when a particle is just outside the box, one further frame of movement may take it inside the box just past the edge, so that the falloff value is slightly greater than zero and therefore it still has some speed, albeit very small. You can see this if you look closely because once inside the box some particles are quicker to get going again than others; they are the ones which by chance moved further into the box thanks to their initial starting position. Try turning off subframe emission in the emitter and you will see that the particles have their speed reduced still further when they arrive at the box and that the particle behaviour is uniform; you can see the same effect by leaving subframe emission on but increasing the number of subframe steps in the project settings.

 The complete animation of this scene can be seen [on the book's website](#).

With these common features dealt with, we'll start by looking at the controller modifiers.

(This page intentionally left blank)

Chapter 16: Modifiers Part 2: Controller Modifiers

The scene file(s) for this chapter can be found in the archive [examples_ch16.zip](#) on the book's website. Click here to [download the archive](#).

16.1 Blend modifier

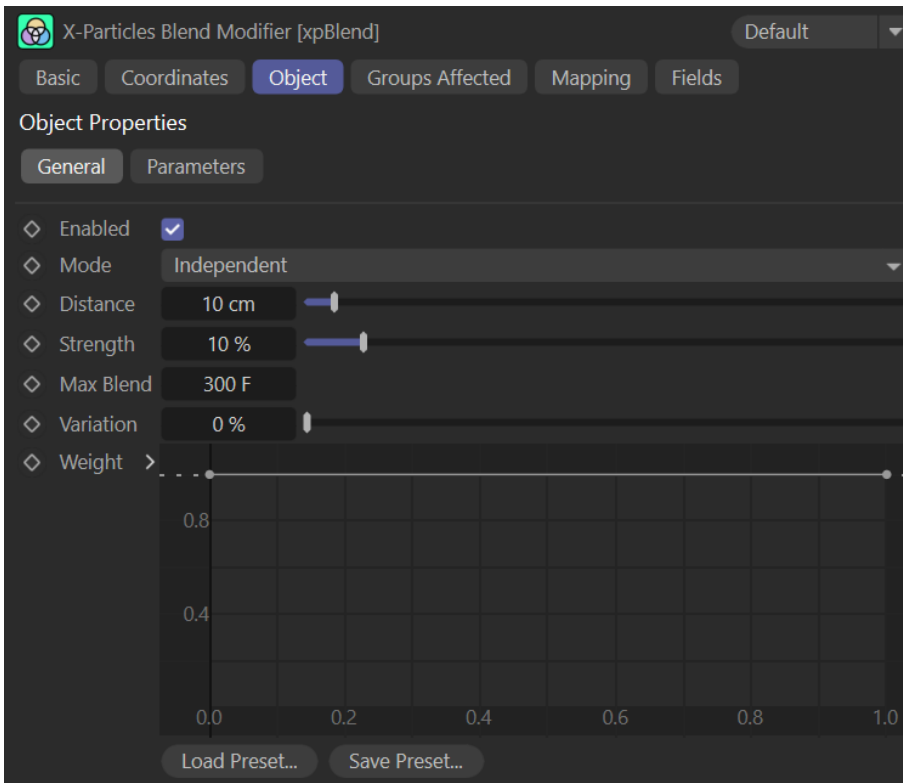


Figure 16.1. Blend modifier

What it does

As the name suggests, this modifier will blend selected parameters of particles which are within a set distance of one another. For each particle, the modifier will blend its chosen parameter(s) with those of all other particles within the specified distance. This can be particles from the same emitter or between different emitters.

Why use it?

Suppose you have a set of particles with varying radius, but you'd like to have them all change the radius to be the same, or at least similar, to each other. You can't use a modifier such as the Scale modifier to do this, because that can only set all particles to a specified value. That would be fine if you knew what value you needed, but often you would not. The Blend modifier effectively averages out the parameter values of neighbouring particles, changing them to be roughly the same as each other.

How to use it

See the example file [ch16_mods_blend.c4d](#). This shows a shot of particles with different radii emitted from a Plane object. The speed is set to zero to make visualisation easier. The modifier is set to blend the radius of those particles which are within 15 scene units of each other. If you play the scene, you can see that clusters of particles which are close to each other change their radius (either up or down) to be similar to one another, but they don't affect particles which are further away, so you end up with small clusters of particles of the same size.

In the modifier, change the '**Distance**' setting to 100 units. On playing the scene you see that all the particles have almost the same size because they are all within 100 units of each other. You can also experiment with blending the colour at different distance settings to see what results you get.

The '**General**' quicktab controls how the blending occurs. The '**Distance**' setting we have already seen. If you change the

‘Strength’ setting you will alter the speed at which blending takes place. In the example scene, with a distance of 100 units and strength of 100%, the particles will all have the same radius after about 180 frames. But if you reduce the strength to 50%, the same result will only be achieved at around 320 frames.

The strength setting applies equally to all particles which are within the specified **‘Distance’** of each other. But what if you would like the strength to vary depending on how close the particles are together? This is what the **‘Weight’** splines does. The default spline doesn’t have any effect but suppose you have a spline like this:

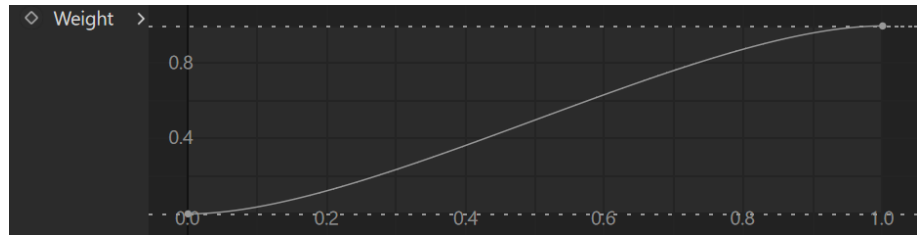


Figure 16.2. Blend modifier weight spline

The way it works is that the value from the **‘Strength’** setting is multiplied by a value between 0 and 1 from the spline. The left-hand end of the spline returns its value when the distance between the particles is zero; from the right-hand end, when the distance between the particles is the same as the **‘Distance’** setting. You can see that with the spline in the above screenshot, the strength of the blend will be greatest when the particles are furthest apart (but still within the **‘Distance’** value) and lowest when they are closest together.

The **‘Max Blend’** setting controls how long the modifier will continue to blend the parameters. In the same scene file, with distance at 100 units and a strength of 100%, if you reduce **‘Max Blend’** to 30 frames, you can see that blending stops after 30 frames have elapsed. This is a per-particle setting and the count starts when a particle enters the modifier field of effect. You can add some variation to this setting by setting **‘Variation’** to more than zero.

The **‘Parameters’** quicktab is very simple and just contains switches for the various parameters to be blended. You can select any of these, alone or in combination.

16.2 Change Group modifier

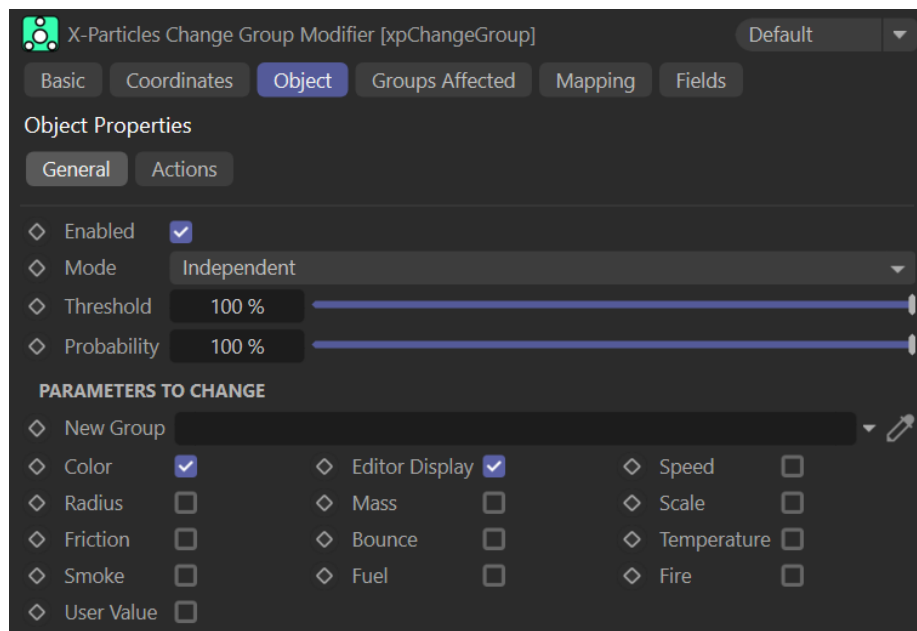


Figure 16.3. Change Group modifier

What it does

This modifier changes a particle’s group. A particle can only be in one group at a time, which is simply an integer value – the group number – held in the particle data.

It’s important to realise that changing the group does not change any other particle characteristics unless you want it to. A particle

group can specify a lot of data - colour, shape, speed, radius, and scale, plus optionally physical data such as temperature, mass and so on. When the group is changed none of this data is used to change the existing characteristics of the particle. However, this modifier can do that as well. In the **'Parameters to Change'** section there are a number of switches. Turning on any of these switches will set the corresponding particle data from the new group. By default only colour and editor display shape are turned on, but you can add any others or turn these off if you wish.

Finally, it is also possible to add Action objects to the modifier which will be triggered to work on a particle if the particle's group is changed.

Why change the group?

As we have seen in previous chapters, a particle's group can influence:

- which modifiers affect a particle
- whether an action affects a particle
- if a particle collides with a scene object or other particles
- whether various dynamic objects such as constraints, FLIP fluids, etc. affect a particle
- if a question object which tests which group a particle is in will pass or fail
- which particles have generated objects such as sprites or trails associated with them

So, there are a lot of reasons why a particle's group can be very important, which makes being able to change the group extremely useful.

How to use it

The example file [cb16_mods_chgggroup.c4d](#) shows various features in action. The emitter only emits particles in group 1 (blue circles). When they pass through the modifier some of them change to group 2 (red pyramids) which are also slightly larger and quite a bit faster. The cube object has a Collider tag which has its own **'Groups Affected'** list which contains only group 2, meaning that only group 2 particles will collide with it. The final frame is shown in Figure 16.3.

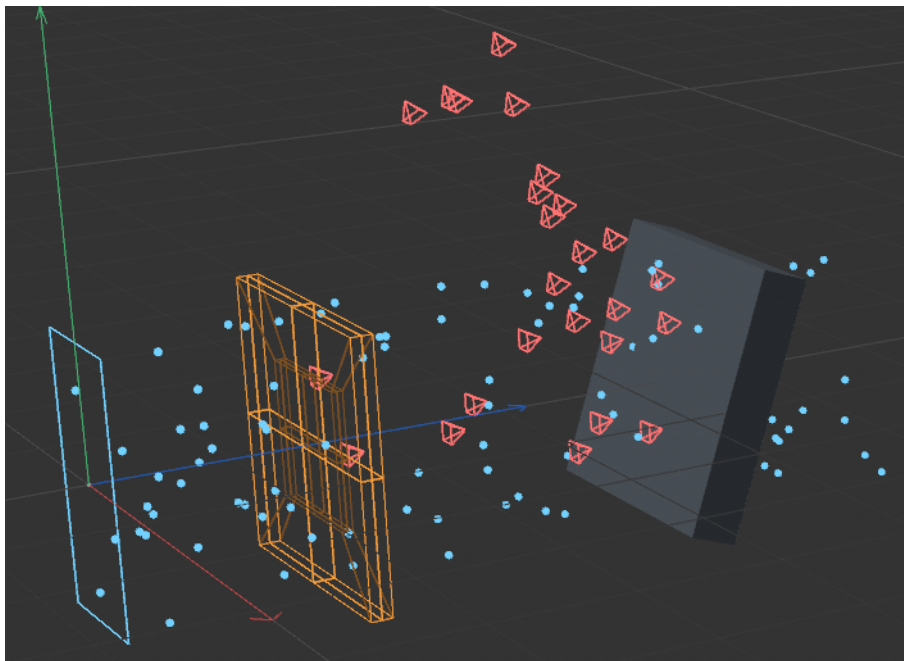


Figure 16.3. Changing a particle's group

The file demonstrates changing the particle radius and speed values to those of the new group. It also shows the control you have over whether a particle actually changes group or not. The first control is the **'Threshold'** setting. If this is set to 100%, the group will only change if the particle enters the inner zone of the field where the falloff value returned is 1.0. Anything less than that, and the group will not change, so those particles which never enter the inner zone will never change group. If you reduce the threshold value, the group will change at lower falloff values as well.

The other setting is **'Probability'**. This simply compares a number randomly generated by each particle with the probability value. The higher this value, the more likely it is that the group will change; a value of 100% means that it will always change, 0% and it

never will. In this scene it is set to 50% which explains why only some of the particles passing through even the inner box of the modifier change their group. It is also why the modifier field/falloff is so 'thin'. Each frame that the particle is in the field of effect of the modifier it is retested against the probability setting, so eventually pure chance determines that almost all particles will pass the test and change group. Try enlarging the field along the Z axis and see what happens.

16.3 Color modifier

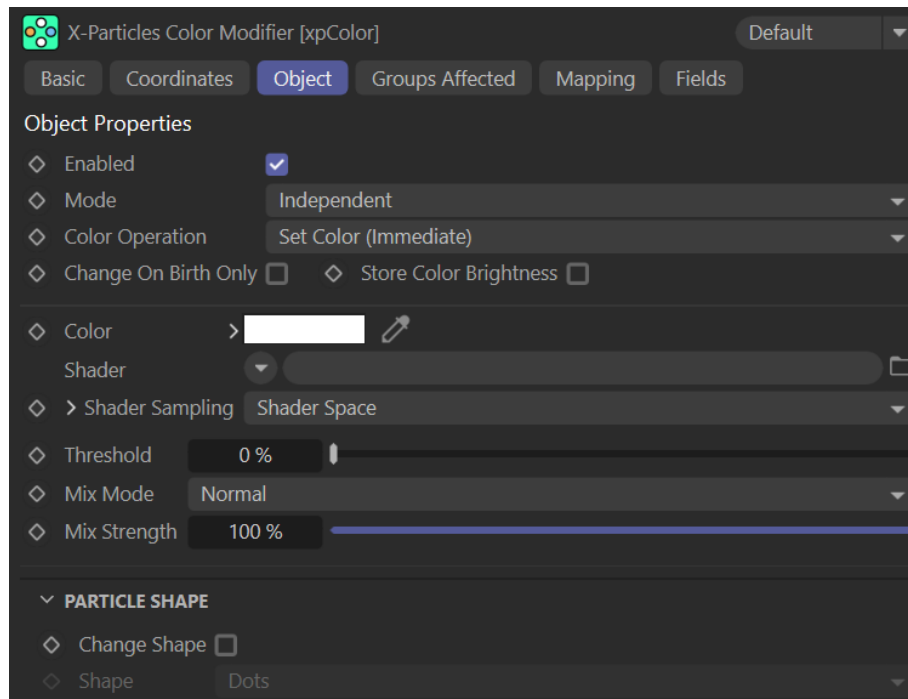


Figure 16.4. Color modifier

What it does

The Color modifier changes the particle colour. This sounds simple, but there are a lot of ways in which you can alter the colour using this modifier.

Why use it?

Well, if you want to change the colour of the particle, this offers the most flexible way of doing it. You can change the colour using settings in the emitter, for example if you use a colour gradient linked to a parameter such as particle age, or you could change the particle's group to one with a different colour, or you could use a question and action to do it. You could even use Python or Xpresso, but the simplest and most powerful way is to use this modifier.

How to use it

The core of the modifier is the setting '**Color Operation**'. This menu controls how the colour will be changed. Depending on what operation is chosen, various other settings will become available.

1. Set Color (Immediate)

This is the default operation. As the name implies, this will change the colour as soon as the particle enters the modifier field of effect (the value returned from a Field object or falloff does not affect this). There are some settings which you can use to change the effect, however.

The switch '**Change on Birth Only**' is a slight misnomer. What it does, if it is turned on, is ensure that the colour is only changed by the first modifier the particle encounters. See the example file [ch16_mods_color_birthonly.c4d](#). This has two Color modifiers; if you play the scene, the first modifier turns the particles red, the second purple. Now turn on this switch in the red modifier. Note that the effect is the same; nothing appears to have changed. Next, turn on the switch in the purple modifier. Now you see that the particles turn red, but not purple. What this switch does is to ensure that only the first modifier changes the colour, any others will not do so (if this switch is turned on in them). The main reason for this is to cooperate with the Infectio modifier discussed in section 16.7 below.


'Store Color Brightness' is another switch designed to help the Color modifier work with the Infectio modifier, and is discussed in section 16.7.

The **'Color'** field lets you specify the new colour. But you can also set the colour from a channel shader which you add to the **'Shader'** link field. This can be any channel shader but some give better results than others. Some shaders can be sampled in 3D (Z axis as well as X and Y) some only 2D. 'Noise' is a 3D shader, while 'Checkerboard' is 2D.

The results you get from using a shader (or bitmap) depend on a number of other settings in the modifier. The first is **'Shader Sampling'**. If you look at the example file [ch16_mods_color_shader.c4d](#) you see that the shader is a red-green checkerboard. The sampling is **'Shader Space'** (the default setting) and the shader is sampled from the particle position in the 3D world. Because Checkerboard is a 2D shader, the particle colour won't change as the particle passes through the field of effect. If you change the shader to a Noise shader, however, you will see the colour change constantly as particles pass through the field of effect, because Noise is a 3D shader.

You can alter this by changing to a random sample, either fixed or variable. If the sampling is set to **'Random (Variable)'** the sampling is done at a different random position each frame. Even with the 2D Checkerboard shader, this will change the colour randomly each frame. However, if set to **'Random (Fixed)'** then the colour is set from a random position but that position is constant for each particle. This sets the colour randomly once, but after that it never changes. Try this with the example file to see the different effects.

If you use a shader, there are several other settings you can use to adjust the result. To see these, expand the **'Shader Sampling'** section by clicking the little black arrow next to the label. The scale settings work as the name implies, they scale the shader or bitmap along the respective axis. The offset parameters move the shader or bitmap along the chosen axis; you can use this to centre a tiled bitmap, for example. The mirror switches simply flip the shader/bitmap around the selected axis. Finally, if **'Tile'** is turned on and the shader is a bitmap, changing the scale will cause the bitmap to be tiled along the scaled axis. If **'Tile'** is off, the bitmap will be scaled without tiling it.

 The **'Tile'** switch only affects bitmaps and has no effect on procedural shaders.

The **'Threshold'** setting adds a degree of time delay to the change of colour. In the example file [ch16_mods_color_threshold.c4d](#), you can see that the colour change takes place as soon as the particle enters the field of effect. If you increase the threshold setting to a high level (say 90%) you will see that not all particles change at once; the colour change appears to blend in slowly. What is actually happening is that each frame each particle generates a random number from zero to one. If this number exceeds the value in the threshold setting (which although expressed as a percentage actually returns a value between 0 and 1) then the particle will change colour, otherwise it will not - but it may do on the next frame. The higher the threshold value, the less chance there is of the colour change taking place in any given frame, but if the particle remains in the field of effect for long enough, it will eventually change. If you play this scene and watch carefully, you can see that a few particles don't change colour before the particles leave the field of effect.

Finally, if you use a shader, the colours from the sampled shader and the **'Color'** field can be combined. To see the effects, in the file [ch16_mods_color_shader.c4d](#) try setting the **'Color'** field to pure blue, leaving the checkerboard at red and green. The **'Mix Mode'** and **'Mix Strength'** settings determine how the colours are mixed. The default is **'Normal'** mode and a strength of 100%. This will cause the shader colour to completely replace the **'Color'** field colour. A strength of zero has the opposite effect: the shader colour is ignored and the **'Color'** field is used instead. Values between zero and 100% will mix the two colours.

The mode **'Add'** will add the two colours together. In the sample file, the red squares are added to the blue colour and the result is magenta; the green square become cyan. **'Multiply'** multiplies the colours together and the result is that all the particles turn black. This may seem odd, but consider what actually happens. The shader colour is red or green. Internally, colours are represented by three numbers between zero and one, one each for the red, green and blue components. A pure red colour therefore has the value 1 for red and zero for green and blue. Pure blue on the other hand is zero for red and green, and 1 for blue. When multiplying colours, the three components are multiplied separately, so red (1,0,0) multiplied by blue (0,0,1) gives a final result of 0,0,0 - which is black. The same happens with the green squares.

'Subtract' is also slightly odd because the result will be different depending on whether the shader colour is subtracted from the **'Color'** field colour or the other way around. The Color modifier always subtracts the **'Color'** field colour from the shader sampled colour. This may help to understand the results you see.

2. Set Color

This mode works in exactly the same way as **'Set Color (Immediate)'** with one important difference. **'Set Color (Immediate)'** changes the colour as soon as the particle enters the modifier's field of effect, regardless of the settings of any Field object (or falloff pre-R20). **'Set Color'** on the other hand is affected by the value returned from the Field/falloff so that the colour change may be a more gradual transition. See the example file [ch16_mods_color_setcolor.c4d](#) to see how the falloff value affects the rate of colour change.

3. Fade Between

A very similar mode to **'Set Color'** but which gives you much greater control over the rate of transition between the original particle colour and the new colour. Like **'Set Color'** the rate of change is affected by the Field/falloff value, but you have two additional settings. **'Fade Value'** is the amount by which the colour will change each time a change occurs. Note that this is a percentage change, not an absolute value. By default, this change will occur every frame, so if **'Fade Value'** is set to 5% the time to complete the change will be 20 frames ($5\% \times 20 = 100\%$). You can change this behaviour too, using the **'Change Every'** setting. By increasing this to 5 frames and the fade value to 20%, you will get the same colour change in the same time but it will occur in five distinct steps rather than a continuous fade. See the file [ch16_mods_color_fadebetween.c4d](#) for an example.

4. Set by Falloff

This is an interesting mode. In the previous modes the colour change is permanent once the particle enters the field of effect. This mode acts just like **'Set Color'** mode and the value from the Field/falloff affects the rate of colour change. However, if you look at the example file [ch16_mods_color_setcolor.c4d](#), the particles eventually turn red and they keep their red colour - it doesn't revert back to blue as the particles move out of the falloff area. In that file, change the mode to **'Set by Falloff'**. Now you see the particles turn red as they move into the field of effect but then turn back to blue as they move out of it.

5. Increment/Decrement Color

In this mode you don't change the particles to a new colour you choose. Instead you increase or decrease the value of each of the three colour components. In the example file [ch16_mods_color_incddec.c4d](#) the red component is increased by 5 each frame while the blue component is decreased by 5. The green component is unchanged. The effect is to change the colour so that the red is at maximum, there is no blue component, and the green is unchanged - which gives a yellow color. If the particles were all of different colours though, each would be changed in the same way and they would each end up a different colour.

6. Random

The modifier in this mode simply sets each particle to a randomly-generated colour. If the switch **'Use Fixed Point'** is turned on, the change will take place only once, when the particle enters the field of effect. If it is off, the random colour change will take place each frame.

7. Use Texture Tag

This mode sets the colour by sampling a channel shader in a material. The material is obtained from a texture tag which you drag and drop in the **'Texture Tag'** field. This lets you use an existing material to colour particles, rather than having to duplicate in the modifier what might be a complex shader tree from a material. You can choose from a variety of channels and the one you choose doesn't have to be used in the material itself. In other words, you can enable (for example) the luminance channel in the material, set up the shader in that channel, then disable that channel in the material. Then you would choose 'Luminance' in the **'Channel'** menu in the modifier. The luminance channel will still be sampled even if it's not being used in the material to shade an object. The example file [ch16_mods_color_textag.c4d](#) demonstrates this using an Earth shader in a material on a Plane object.

But where in the material space is the channel sampled? By default it is sampled at a random point which is different for each particle, but once generated the point will be constant for that particle. The sampled colour will therefore never change for each particle unless the sampled shader is animated. However, there is another option. If the particle has been emitted from a polygon object, the emission point will have UV coordinates on the object. You can store these coordinates by turning on the switch **'UV Emission Data'** in the emitter's extended data tab. If that data is available, the modifier will use those coordinates as the sample point.

There are two important caveats with this mode. The channel you sample must have a bitmap or procedural shader in the **'Texture'** field in the channel. For example, in the material Color channel you can't just set the colour and leave the **'Texture'** field empty; you would have to add a shader, such as a Color shader, into the texture field. The second issue is that, although one of the available channels is 'Reflection' this will only work with the old-style reflection channel and not the new 'Reflectance' channel which appeared in R18.

8. Use Vertex Color Tag

This is similar to using a texture tag, but instead you supply a vertex color tag (available in R18 onwards). In addition, the particle must be emitted from the vertices of a polygon object, and in the emitter's extended data tab the switch **'Emission Vertex'** must be turned on.

9. Gradient by Parameter

This is very similar to using the **'Gradient (Parameter)'** colour mode in the emitter's display tab. See Chapter 6, section 6.1.3 for more details on how this works. Not all the parameters available in the emitter are used in the modifier, and there is no 'auto' switch to calculate minimum and maximum values - you will have to supply these yourself.

There is one parameter not present in the emitter and this is **'3D Linear'**. This parameter uses the particle position along one of the orthogonal 3D axes, X, Y or Z. This is not how far the particle has moved along the axis, it is the particle position in the 3D world. When the position along the axis is less than **'Axis Min'** the colour at the left of the gradient is used; when the position exceeds **'Axis Max'** the colour at the right of the gradient is used. The colour change will start when the position is greater than **'Axis Min'** and will be complete when it is equal to or greater than **'Axis Max'**.

See the example file [ch16_mods_color_3dlinear.c4d](#). The particle moves along a circle spline and the axis is set to the Y axis (set in the **'Linear Mode'** menu), with the default blue to white gradient. As the particle moves up along the circle, it reaches the top of the circle which is at Y = 200; at that point, the position is equal to 'Axis Max' and the colour change to white is complete. As it continues to move along the spline the Y position decreases and the colour changes back to blue. Note that when the Y position becomes negative, it is less than **'Axis Min'** (zero) and so remains blue until the Y position exceeds zero again.

The other setting here is **'Space'**. If set to **'World'** the position is the actual particle position in the 3D world. If it is set to **'Local'** and an object is present in the **'Object'** field, the **'Axis Min'** and **'Axis Max'** values are offset by the position of the object. In the example file, try moving the circle spline from the world centre to a Y position of 100 units. You will see that now the particle becomes white before it reaches the top of the circle, as expected because the top of the circle is now at Y = 300. But if you change the space to local and drag the circle spline into the object field, the particle changes to white at the top of the circle again.


10. Time-Dependent

In this mode the colour change takes place over a predetermined time period. This starts when the particle enters the modifier field of effect, at which point the particle is given the colour at the left of the gradient. It is complete when the time given in **'Time to Completion'** has elapsed, and the particle now has the colour at the right of the gradient. You can try out this mode with the example file [ch16_mods_color_timedep.c4d](#).

There is an important setting in this mode. The menu **'On Completion'** controls what will happen when the colour change is complete. The default is **'Do Nothing'** so the particle colour remains that from the right of the gradient (red in the example file). The next option is **'Wrap to Start'** which will immediately set the colour back to the left of the gradient (blue) then repeat the colour change over the next time period. Finally, **'Reverse'** will change the colour back to the left of the gradient over the next time period. In both cases, the colour changes will repeat as long as the particle is within the modifier's field of effect.

11. Distance From Object

Here you supply an object by dropping it into the **'Object'** link field and the distance of the particle from the object is tested. Any particle more than **'Furthest Distance'** away from the object will have the colour at the right of the gradient. As the particle moves closer to the object, it will change colour until it is equal to or less than the **'Nearest Distance'** value, at which point it will have the colour from the left of the gradient. If it moves further away from the object again, the colour will move towards the right of the gradient.

 Note that the **'On Completion'** menu is still available in this mode. In fact, this is an interface error and that menu has no effect in this mode.

12. Distance From Camera

This works in the same way as **'Distance From Object'** but uses the current camera. If you haven't added a camera to the scene, the default camera is used. The same caveat regarding the **'On Completion'** menu applies here as it does in **'Distance From Object'**.

13. None

With this option the modifier will not change the colour at all. The reason this option exists is so that you can change the particle shape using the final option in the modifier without altering the colour.

14. Change Shape

This option lets you change the particle shape very simply by using a modifier. Otherwise, to change the shape you would have to do something such as change its group or use an Action, both of which require more work to set up. To change the shape, turn on this switch and select the desired shape from the **'Shape'** menu. This can be done without changing the colour by choosing **'None'** from the **'Color Operation'** menu.

16.4 Custom Data modifier

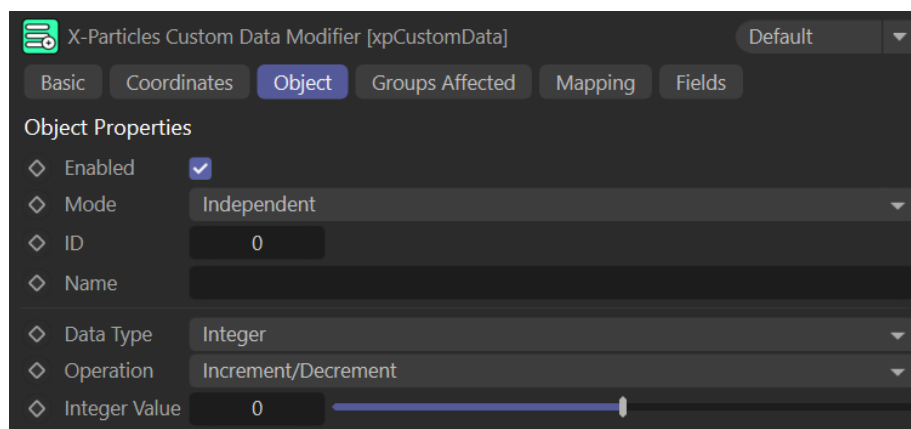


Figure 16.5. Custom Data modifier

What it does

The nature of custom data is discussed in Chapter 5, section 5.5 'Custom Data'. Once you have added custom data items it's very useful to change their value, which is what this modifier does.

Why change the data?

Please refer to Chapter 5, section 5.5 for a detailed example, which shows what you can do by changing and testing the values of custom data items.

How to use it

This is very easy. The first thing you have to do is identify which data item you want to change. If you don't specify the correct data item, or it doesn't exist, this modifier will do nothing. You can identify the item either by entering its ID number into the **'ID'** field and/or entering its name into the **'Name'** field. Then you need to specify the type of data it is. This must match the type of data you entered when creating the custom data in the emitter. This is crucial because you can't (for example) enter a vector value into a custom data item which is an integer, so if the data types do not match, the modifier will do nothing. When you set the data type there is a value field which will change name to reflect that type, so a data type of integer will name the field **'Integer Value'**, a vector data type will display a **'Vector Value'** field and so on.

Next, you specify what you want to do in the **'Operation'** menu. You can choose to take the existing value of the data and add the value field to it. This is the default and is the most useful; to do this, **'Operation'** is set to **'Increment/Decrement'**. For example, Figure 16.6 shows a vector data item which will take the current value then do nothing to the X component, subtract 1 from the Y component and add 2 to the Z component.

Alternatively, you can set **'Operation'** to **'Set Value'** and then the data item's value is set to that in the value field.

That's all there is to this modifier. All it does is let you change the custom data's value; the real power comes when you test this value and take action accordingly, as explained in section 5.5. of this book.

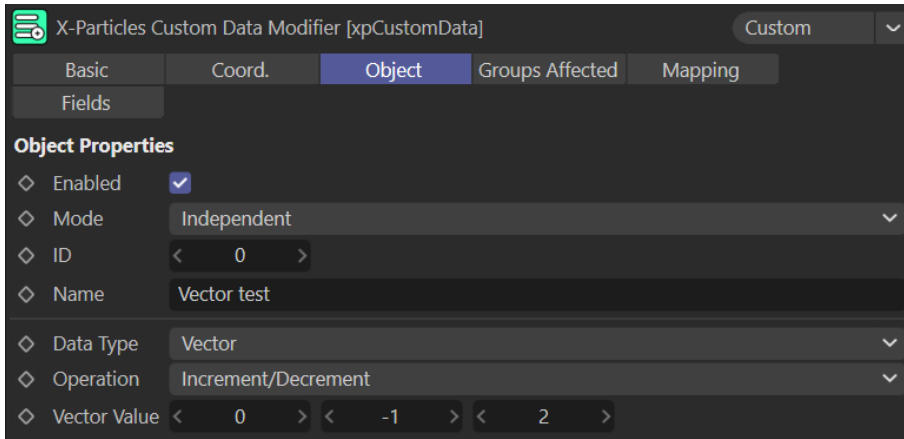


Figure 16.6. Changing a custom data item of vector type

16.5 Freeze modifier

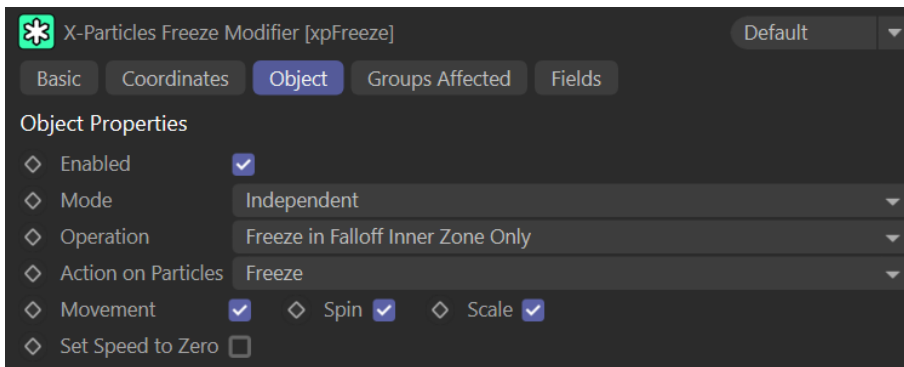


Figure 16.7. Freeze modifier

What it does

This very simple modifier freezes particles (or can unfreeze them if they are frozen). If you need to bring a particle to a halt, you could of course simply set the speed to zero. Freeze doesn't work quite like that. What it does is set flags in the particle telling the emitter not to move it. The speed, however, remains set in the particle and if it is unfrozen it will carry on moving with the speed and direction it had before it was frozen.

Why freeze a particle?

You might want to do this to bring a temporary halt to particle movement but with the option of resuming that movement later on. Also, with this modifier you can opt to freeze three possible motion changes in the particle. You can just freeze actual movement, but leave spin and changes in scale (or radius) unfrozen, or make any combination of these changes.

How to use it

Most of the time you will use this to freeze particles, but you can unfreeze them with it instead. For example, you might use an animated Freeze modifier to move across a particle stream and freeze the ones in its field of effect. Then you could another modifier to do the same thing but unfreeze them. The example file [ch15_mods_unfreeze.c4d](#) demonstrates this.

An important setting is the **'Operation'** menu. The default is **'Freeze in Falloff Inner Zone Only'**. Freezing a particle is an on or off effect. With this mode, the particle will only be frozen if it enters the inner zone of the field (or falloff) where the returned falloff value is 1.0. The other choice is **'Freeze Depends on Falloff'**. A particle entering the inner zone will always be frozen but in the outer zone (still within the field of effect, of course) the probability of it freezing is lower the further away from the inner zone the particle is. Note that this is dependent on a random number generation and is retested each frame, so if the particle remains in the outer zone long enough, it will probably freeze at some point. The file [ch15_mods_freeze_falloff.c4d](#) demonstrates this. Play the scene and note that only the particles which enter the inner zone freeze. Now change the operation mode and you see that many particles freeze soon after entering the field, some take longer, and a few escape freezing altogether.

Apart from that, the only other options are which types of motion change you want to freeze. By default the modifier freezes all three but you can turn off any of the three switches to prevent freezing (or unfreezing) a specific motion type. If you freeze movement, you also have the option to set the particle speed to zero when it is frozen. That's just for convenience in case you really do want to zero the speed.

The example file [ch15_mods_freeze.c4d](#) demonstrates different freeze types. The particle is moving, spinning (thanks to a Spin modifier) and increasing in size (due to a Scale modifier). When it enters the field of effect of the Freeze modifier, all three motions are halted. You can try turning off one or more switches to see what happens.

There is one thing to remember when using this modifier. Once the particle enters the modifier it freezes but also remains in its area of effect unless you move the modifier. So if you try to use an action to unfreeze the particle, it won't work because the particle is still in the modifier's field of effect and will immediately be frozen again. The way to solve this is either to move the modifier away from the particle, then unfreeze it, or (a better option because it gives you more control) is to use the modifier in action-controlled mode. You can then turn the modifier on when the particle is created, then when you want to unfreeze it, turn the modifier off and unfreeze the particle with an action.

16.6 History modifier

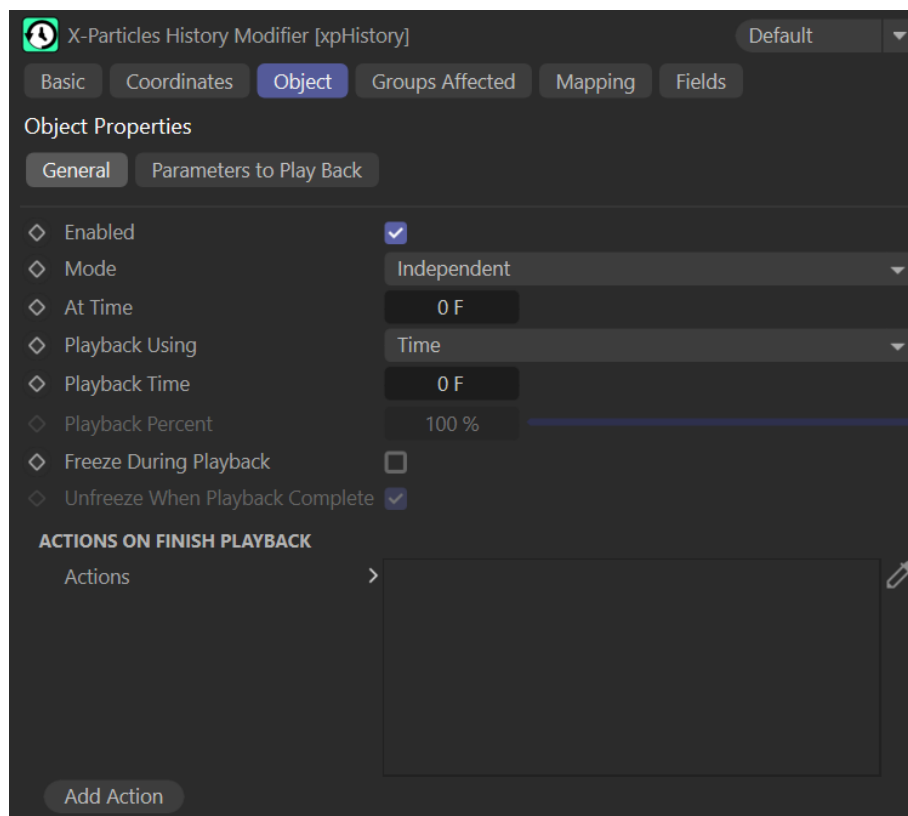


Figure 16.8. History modifier

What it does

The History modifier stores a set of particle parameters in memory then at a specified time plays them back. For example, if particle position is played back, then from the time the playback starts the particles will appear to retrace their previous steps.

Why use it?

Animations where objects move in various directions but then the animation appears to reverse itself and run backwards are common, and you can duplicate that effect with this modifier. However, you aren't restricted to making particles move in reverse; if other parameters such as colour, radius, rotation, etc. change during the animation, these can be played back too.

See the example file [ch16_mods_history.c4d](#). Particles are moved by a Turbulence modifier, and at frame 60 the playback starts. Playback lasts for 60 frames so the particles move back to their start position. At frame 120 the playback comes to an end and the

Turbulence modifier takes over moving the particles again. But it isn't only the particle position which is played back here; the particle colour is changed to red by a Color modifier, and when playback occurs the colour is wound back from red to the original blue. The colour playback can be disabled in the **'Parameters to Play Back'** quicktab by turning off the **'Color'** switch, if you wanted to retain the red colour. Finally, there is the option to trigger one or more Actions when the playback is complete, as is done in this scene to change the particles to yellow when playback is over.

How to use it

First, note that you don't have to do anything to start the modifier recording particle data. As long as it is enabled it will do that as soon as the animation starts, both in independent and action-controlled mode. To use the modifier to playback the recorded history, you need to do three things:

- set when playback will start
- set for how long it lasts
- decide which particle parameters you want to play back

To set the start of playback, change the **'At Time'** setting. This is the scene time when playback commences. Note that that modifier can only carry out one playback session; you can't set it to playback more than once starting at different frames.

Once playback has started how long will it last for? The maximum possible length of any playback is, of course, the **'At Time'** setting minus the scene start time, which is usually frame zero, but doesn't have to be. Be aware of this when the start time is not frame 0. In the same example scene, set the start frame of the scene to -20 frames, then in the emitter set the **'Shot Time'** to frame -19. At first, this looks exactly the same as before when played, but the playback still starts at frame 60 - by which time, 80 frames have elapsed. If the length of playback is still 60 frames, it will be complete at frame zero, at which point the particles won't have returned to their starting position. A way around this problem is to set **'Playback Using'** to **'Percent'** rather than **'Time'**. This will always use the specified percentage of the elapsed time, so in this case if the **'Playback Percent'** is set to 100%, the playback will be complete when the particles return to their starting position and you don't have to calculate how much time is required.

i If the length of playback is set to be greater than the elapsed time in the scene, that doesn't actually matter - the excess frames are just ignored.

Finally, you must decide which parameters to play back. There is a list of them in the **'Parameters to Play Back'** quicktab. It doesn't matter if you leave switches turned on for unused parameters. If the particles in your scene do not rotate, for example, and you leave the **'Rotation'** switch turned on, that will have no effect since there is no rotation to play back.

There are two other options you can use. There is a switch named **'Freeze During Playback'**. Look at example scene [ch16_mods_history_freeze.c4d](#). This uses an X-Particles Fragmenter object to generate the faces of a pyramid and move each face by moving a particle. If you play this scene and when you get to about frame 55 stop it and then play it one frame at a time, you can see that the faces never move completely back to their starting position - a slight gap is left between the faces. This is due to how the emitter works during playback. What happens is that in each playback frame the History modifier moves the particle back one step along its original path, but then the emitter moves the particle independently. Most of the time this is never apparent when watching the scene, but using a Fragmenter is one case where it may be very noticeable.

The only way to solve this is to stop the emitter moving the particles. We could do this with a Freeze modifier or Freeze Particles action, and you could amend this scene to add a Question object. This would test the particle age and when that was 30 frames (i.e. when playback starts) it would trigger a Freeze Particles action. This works perfectly well and if desired you could add another action to unfreeze the particles when the playback was complete.

This involves some extra work so there is a much simpler way: turn on the **'Freeze During Playback'** switch. This has the exact same effect as the use of a freeze action or modifier, and in fact uses the same internal particle flags. If you want the emitter to move the particles again when playback is complete, turn on the **'Unfreeze When Playback Complete'** switch. You can even mix and match these methods; using the switch to turn on freeze and then an action to unfreeze them works fine.

Using the modifier in action-controlled mode

This is explained fully in Chapter 13, section 13.8.3 'Control History'. The advantage of this mode is that instead of playback starting for all particles at the same time, you can control when playback starts for individual particles if necessary. Very importantly, unlike most modifiers, this modifier in action-controlled mode will carry on working and always record the particle data. The action only sets when playback will start and its length.

Other points to note

If using the modifier to make particles retrace their paths, you only need to turn on the **'Position'** parameter. However, if you turn off the **'Speed'** or **'Direction'** parameters, this will affect what happens after playback is over. Try this in the example file [cb16_mods_history.c4d](#) - turn off one or both of those switches and see what happens when the playback is complete. This happens because if you play back speed and direction as well as position, when the playback is complete the particles will not only have the original starting position but their original velocity as well. If you don't play back speed and/or direction, the Turbulence modifier in the scene continues to change the particle velocity during playback, but you won't see any effect of that until the playback is done.

When using a Trail object with this modifier, the playback does not affect the trails at all. In the example file [cb16_mods_history.c4d](#), enable the Trail object in the object manager and watch the playback. The trails are not rewound, only the particles. In fact, it goes further than that. While the particles are being played back the trails are still being generated with additional vertices added each frame in the same positions as existing vertices. This has an unexpected effect if you turn on the **'Freeze During Playback'** switch. Try that and see what happens. The reason for this is that freezing the emitter has the side-effect of preventing further trail points from being generated. Consequently, when the particles are unfrozen at the end of playback the spline is connected from the last trail vertex (where the particle was when playback started) to the new vertex (where the particle is now that playback is complete). There is no way to prevent that from happening, since frozen particles will never generate trail points.

16.7 Infectio modifier

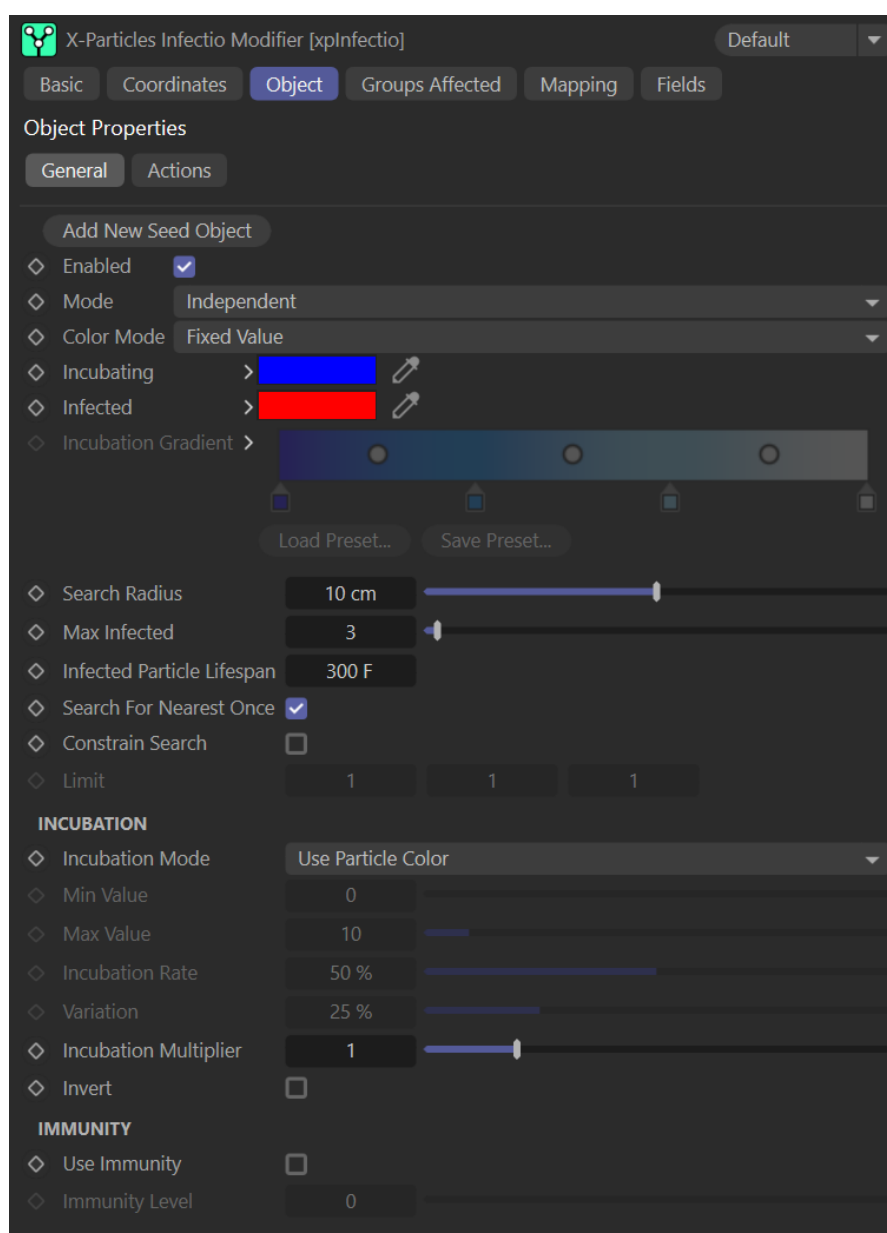


Figure 16.9. Infectio modifier

What it does

Infectio produces a simple simulation of the spread of infection in a population, although you can use it for other things too. It doesn't generate any additional particles - these are generated in the usual way - but what it does is alter the state of the existing particles.

The way it works is simple. Before the simulation starts, all particles are in the uninfected state. When one or more particles are made infected (at the start by a special seed object used for this purpose) they can infect other particles. Which ones are infected, how many, and how quickly that happens are controlled by the settings in the modifier. Particles to be infected are first placed into an incubating state. In that state they cannot infect other particles but they will eventually turn infected and then start infecting other, uninfected particles.

The speed with which the infection spreads is determined by several factors, but one of the most important is the incubation rate. The higher this is, the faster infection spreads, and there are several ways in which this rate can be set.

To be clear, particles can be in one of three states:

- uninfected
- incubating infection: these will not affect other particles but with time will change to the infected state
- infected: these particles will infect uninfected particles, changing their state to incubating

See the example file [cb16_mods_infectio_basic.c4d](#). This shows the initial uninfected particles (coloured green) emitted from a Plane object using a Noise shader. An infected particle is introduced by a Seed object, and then more particles become incubating (blue particles) and these in turn become infected (red particles) which proceed to infect more particles. We'll use this file as a reference file to explore the many settings in the modifier. Note how not all particles become infected. Those which are too far away from an infected particle won't become infected so little islands of uninfected green particles are left.

Why use it?

Essentially, the modifier provides an interesting simulation which you can experiment with if you want to see how infection might spread in a community. But that's the highbrow answer. In most cases, it's just fun to play with and can make some interesting patterns. This image was generated from an Infectio simulation by using a Trail object to connect infected particles:

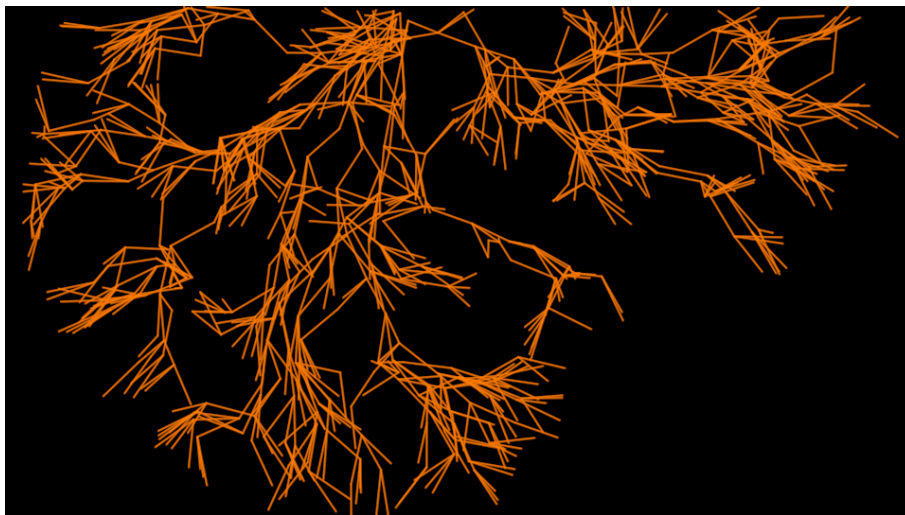


Figure 16.10. Trails generated by Infectio

How to use it

The first thing you will need are some particles. These can be generated any way you like, but for experimenting with the modifier it's easier to do as the example file does: generate a single shot of particles and give them a speed of zero. It can be quite interesting to generate particles using the '**Rate**' option in the emitter and give the infected particles a short lifespan. You can see this in the example file [cb16_mods_infectio_rate.c4d](#). Note how the infected particles in an area can die out completely and are replaced with uninfected particles, while in other areas the new uninfected particles are re-infected with what looks like another 'wave' of infection.

The second step is to add a Seed object to generate some infected particles to kick start the simulation. You can add as many Seed

objects as you like using the **'Add New Seed Object'** button in the modifier. Infection will be introduced into the population at each seed point (assuming there are uninfected particles within the seed's area of effect). The Seed object is discussed more fully below.

That's all you need to do for a basic simulation. If nothing happens when you run it, the likelihood is that no uninfected particles are in the Seed object's field of effect; you can either move the Seed object to another location or increase the size of its field to include some particles. You can also animate the Seed object's position so that at some point it finds some uninfected particles to work on.

There are now a lot of settings you can change to alter the simulation.

1. Colours

Colour is important in this modifier for two reasons. Firstly, a colour change can be used to show when a particle changes state to incubating or infected. Secondly, the initial colour of the particle (that is, the uninfected colour) can be used to control the incubation rate.

Infectio can set the colour of incubating and infected particles in three ways. The default is to use two fixed colours. Particles will change colour immediately when they are in the incubating state and again when they are infected. Alternatively, you can use a colour gradient. Particles which change to incubating have the colour at the left of the gradient then change colour as they incubate, until finally when they become infected they have the colour at the right of the gradient. You can test this in the reference file by changing **'Color Mode'** to **'Gradient'** but in this scene the change happens quickly because the incubation rate is high. A third method is to change the particle group. You need to specify a group for incubating particles and another for infected ones. The advantage of doing this is that you can change other particle parameters at the same time, such as speed, radius, mass, particle shape and so on as well as colour.

2. Search options

Infected particles search their surroundings for more uninfected particles to move to the incubating state. Each infected particle searches within a radius you specify. With the reference file you can see that small pockets of particles remain uninfected because the distance from the nearest infected particle is too great. If you experiment with the **'Search Radius'** setting, you can see that it has a profound effect on the simulation.

Should infected particles search just once or in every frame? That is what the switch **'Search for Nearest Once'** controls. If this is turned on, each infected particle will look just once for others to infect, but if it is off, they will look for more particles each frame. If you use a particle shot as in the reference file, this makes no difference because no new particles appear as the animation runs, but if you switch to rate emission this has a significant effect.

Infected particles can be set to have a short lifespan. This is controlled by the **'Infected Particle Lifespan'** setting. In the reference file, infected particles live as long as the scene so eventually most particles are red but the number is unchanged. If you reduce that setting to a small value - say, 20 frames - it doesn't affect the simulation but at the end most particles have died leaving only a few uninfected ones. Again, this setting makes a much greater difference if new particles are constantly being generated. You can see this in the reference file by changing the emission mode in the emitter to rate rather than shot. New particles are now being generated but are quickly infected by the infected particles already present and which are not being removed from the scene. Now change the infected particle lifespan to something very short, such as 10 frames. Here you see that large pockets of uninfected particles appear, though they are sometimes overcome by waves of new infected particles from the Seed object. You can also use this modified scene to test the effect of the **'Search for Nearest Once'** switch. Turning this off has a major effect, even if the infected lifespan is very short.

Another question is how many uninfected particles should each infected particle change to the incubating state? This can be changed with the **'Max Infected'** setting. By default each infected particle will convert three uninfected ones, assuming they are within the search radius. In the reference file, try setting this to one - the simulation proceeds as before but is much slower. However, note that increasing the value to five or higher doesn't really do very much. This is because in this scene most infected particles won't have more than three uninfected particles within its search radius.

The final search options are to constrain the search. In the reference file turn on the **'Constrain Search'** switch. The Z component value in the **'Limit Search'** setting is set to 0.2. The smaller each component is, the more the search is restricted along that axis; you can see the effect that this has quite clearly.

3. Incubation

The incubation rate determines how fast particles will convert from the incubating to infected state; the slower it is, the longer they will take. You can assign an incubation rate in various ways.

i Whatever method you choose, you can always adjust the rate up or down as required by using the **'Incubation Multiplier'** setting. If this is less than one, the rate is slowed; more than one and the rate is increased.

The default method to set the rate is to use the particle colour. This is the original colour of the particle, not the incubating or infected colours. The overall brightness of the colour controls the incubation rate, so if the colour is black (the brightness is zero) the particle will never change from incubating to infected. Clearly, if particles have different colours they will have different incubation rates.

You can set a specific incubation rate by choosing **'Set From Incubation Rate'** as the mode. Then set the rate in the **'Incubation Rate'** setting and if desired add some per-particle variation with the **'Variation'** setting.

Next, the rate can be calculated from one of six particle parameters - radius, mass, temperature, smoke, fire and fuel. In each case you must specify minimum and maximum values. For example, when using the radius to determine incubation, if a particle has a radius at or below the **'Min Value'** setting, the incubation rate will be zero, but if the radius is at or higher than the **'Max Value'** the incubation rate will be at its maximum. This means you need to be careful when setting these limits. In the reference file the particles have a radius of 3 units so if you control incubation by radius and set the minimum value to 3, the incubation rate will be zero.

The final control is the **'Invert'** switch. If this is turned on, the incubation rate is inverted. When using radius to control incubation, turning on the switch would mean that the particles with the largest radius would have the lowest incubation rate.

4. Immunity

This feature will make some or all particles immune to infection, and if they are immune they will not even enter the incubating state. To use it, turn on the **'Use Immunity'** switch and set an **'Immunity Level'** of higher than zero.

The way this works is that the immunity level is compared to the incubation rate. If the incubation rate is lower than the immunity level, then the particle is immune. You can see from this that you are likely to see the best results if the particles have different incubation rates since then some particles will be immune and others not. You do need to set the immunity level carefully. If it is too low, no particles may have an incubation rate lower than the immunity level and none will be immune. Set too high, and all particles may be immune and none will become infected.

5. Actions

In the **'Actions'** quicktab you can set one or more actions to be triggered when a particle changes to the incubation state, and other actions when it moves to the infected state. These actions can do whatever you like.

The Seed object

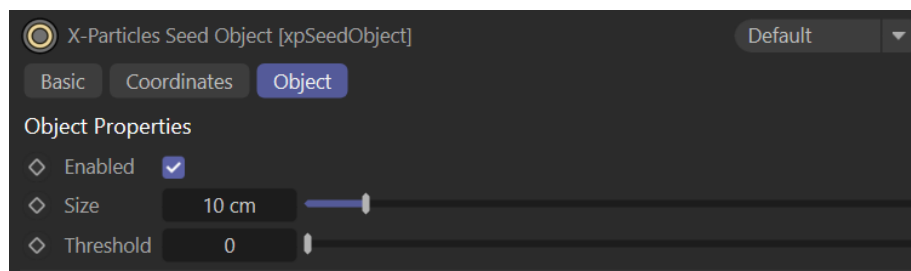


Figure 16.11. The X-Particles Seed object

The X-Particles Seed object is a generic object used whenever you need to specify a starting position for another object. Infectio uses it to control where the simulation should start. You can have as many seeds as you like but to work they must meet these criteria:

- there must be at least one Seed object or nothing will happen
- all seeds used by an Infectio modifier must be child objects of the modifier
- when the simulation starts there must be at least one uninfected particle within the seed's area of effect

The area of effect of the seed is determined by its **'Size'** parameter and is shown in the viewport as a wireframe sphere.

! Note that the seed is always active, not just at the start of the simulation. This means that if you are generating new particles and any of the new particles fall in the area of effect, they will be changed to the incubating state. The only way to prevent that is to

keyframe the **'Enabled'** parameter so that the Seed object is active only in the frames you want.

The other control in the Seed object is **'Threshold'**. This is to solve a problem of particles with very low, or zero, incubation rates being changed to the incubation state but then never changing to the infected state. To prevent that, set this value to higher than zero. Any particle with an incubation rate lower than the threshold will then never be changed by the Seed object to the incubating state, though they may still be so changed by an infected particle.

16.8 Inherit modifier

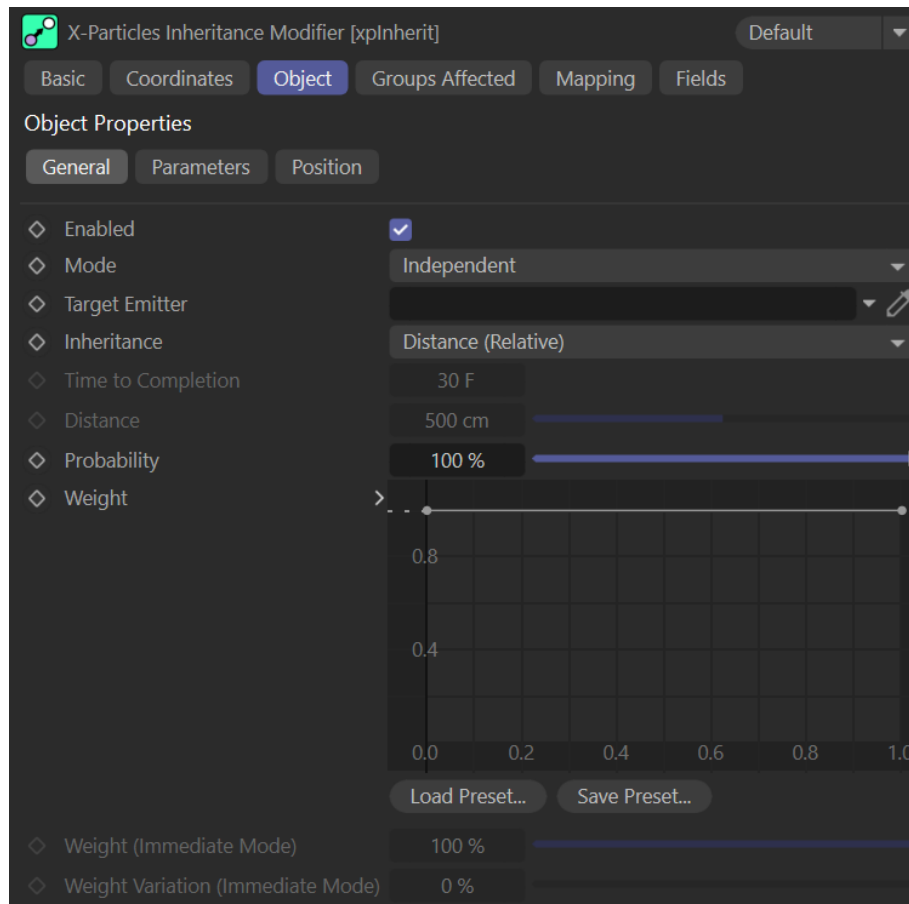


Figure 16.12. Inherit modifier

What it does

The modifier causes the particles from the emitter it affects to inherit parameters from another emitter, which will automatically be set not to be affected by the modifier, since a particle cannot inherit from itself.

A variety of parameters can be inherited, and the speed with which the change in parameter takes place can be adjusted in several ways.

Why use it?

This modifier is useful when you have two emitters - perhaps widely separated in a scene, or doing completely different things - and you want one set of particles to adopt the same parameter(s) as the other set. For example, you might want the colour or rotation of one emitter's particles to change to become the same as the other emitter's particles.

See the example file [ch16_mods_inherit_example.c4d](#). This shows trails being generated from particles emitted from a sphere; you can't see the sphere or the particles because they are deliberately hidden. The other emitter emits one particle for each one in the target emitter, and they inherit the position of the particles from the target emitter. Now enable the 'Inheriting Emitter' in the object manager. The result after 180 frames is that each trail has a yellow particle from the inheriting emitter on its tip and they stay there as long as the scene continues to play. The particles change from yellow to red because they also inherit the colour of the invisible target particles.

How to use it

For this modifier you need at least two emitters. One will be the ‘target emitter’ - this is the emitter whose parameters will be inherited. Particles from the other emitter(s) will inherit the target emitter parameters. In the example files, these will always be named ‘Target Emitter’ and ‘Inheriting Emitter’. The first thing is to drag the target emitter into the ‘Target Emitter’ link field in the modifier. Doing this will ensure that the target emitter will be unaffected by the modifier so that the emitter doesn’t try to inherit from itself.

This raises a question: which target particle does the inheriting particle inherit from? There are several ways in which this could be done, but the modifier uses a very simple algorithm: the inheriting particle inherits from the target particle with the same index in the particle array. Note that this not the target particle with the same unique ID value, but the index in the array, which can change as other particles are deleted and more added. If there are more inheriting particles than there are target particles the modifier chooses an index from the target particle array. If there are more target particles than inheriting particles that won’t matter much of the time, since there will always be a corresponding target particle for each inheriting particle, but it can occasionally cause unexpected effects. See the example file [ch16_mods_inherit_excessparticles.c4d](#). You might wonder why not all the blue particles inherit from the orange particles when there are clearly more than enough orange particles to inherit from. This is because more orange particles are being created than blue ones. By the time 50-60 blue particles are created, the orange particles with the same index have already moved too far away from the blue particle and no inheritance can occur.

i There’s nothing to stop a chain of emitters inheriting from each other but you will need more than one Inherit modifier. You will also need to make sure that each emitter is only affected by the correct Inherit modifier and not the others as well. See the scene file [ch16_mods_inherit_chain.c4d](#) for an example of how this could work.

Inheritance mode

Next, choose the required inheritance mode from the **‘Inheritance’** menu. The simplest is **‘Immediate’** where the inheriting particles immediately inherit the target particle’s parameters. There is no fade in or transition, the change in parameter takes place immediately. Equally straightforward is **‘Time’**. This simply sets a time over which the full change in parameter will occur; you can set this in the **‘Time to Completion’** setting.

A third option is **‘Distance (Absolute)’**. This uses the value from the **‘Distance’** setting. No inheritance will occur if the inheriting and target particles are further apart than this distance. The degree of inheritance depends on the distance between the particles. If the distance is zero, the parameter will be completely inherited; the extent to which the parameter is inherited depends on the distance, so if they move closer together the parameter change in the inheriting particle is greater. If they move apart again, the parameter in the inheriting particle will revert back to its original value. See the file [ch16_mods_inherit_distanceabs.c4d](#) for an example of this.

The final option is **‘Distance (Relative)’**. Instead of using a fixed, absolute difference, a particle will store the distance to the nearest particle when it enters the modifier’s field of effect. That distance is then used in the same way as **‘Distance (Absolute)’**. For any change in a particle parameter to occur, the distance between the particles must be less than this stored distance.

There are additional controls over the change in parameter which are available in all inheritance modes (except **‘Immediate’** mode when the **‘Weight’** control is not available). The **‘Probability’** setting controls whether a parameter change will occur. If it is set to 100%, the change will always occur; if it is zero, no change will ever occur. Between those values the setting is the chance that the change will occur each frame.

The **‘Weight’** control affects the strength of the change in parameter. This can be a little confusing. Suppose the inheritance mode is set to **‘Time’**. In that case, the horizontal axis of the weight spline is the time over which the change takes place. The vertical axis is always a value between zero and one which is multiplied with the change in parameter which is to take place that frame. By default, that value is always 1.0, so the change will be unaffected. If it is less than 1, the degree of change is reduced; if it is zero, no change will take place. By default therefore, the same degree of change will take place in all frames until the parameter has been fully inherited. A spline like this, however, will cause the change to be very small at the start, but then increase very quickly towards the end:

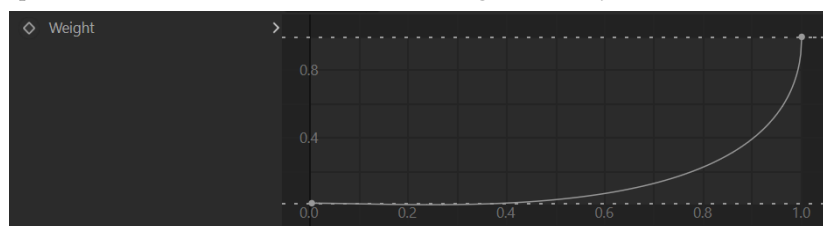


Figure 16.13. Inherit Weight spline

See the example file [ch16_mods_inherit_weight.c4d](#). This uses the above spline to change the inheriting particles' radius and colour over 60 frames. You can see that the radius increases very slowly at first but then very fast as the time nears completion.

With either distance mode, the horizontal axis of the spline is the distance between particles. The spline is not available in **'Immediate'** mode but instead there is the **'Weight (Immediate Mode)'** setting and a variation setting that varies the weight between particles. The spline is not available because the change in parameter occurs immediately, not over time or affected by distance.

Parameters to inherit

The **'Parameters'** quicktab controls which parameters will be inherited. Tick the box for each parameter you want to use. If you turn on the **'Position'** parameter, there are more options in the **'Position'** quicktab.

i To inherit UV data, you must turn on the **'UV Emission Data'** switch in both the target and inheriting emitter(s).

Position options

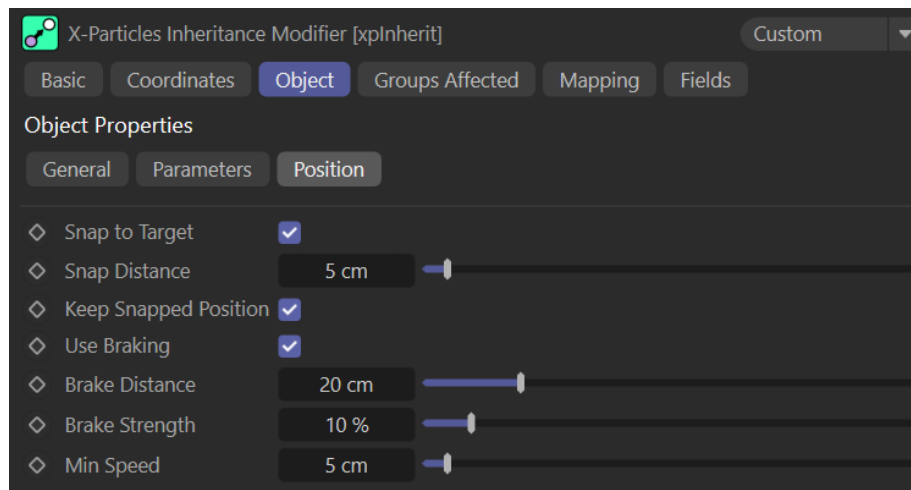


Figure 16.14. Inherit Position options

These are shown in the **'Position'** quicktab, but to use them the **'Position'** parameter must be turned on the **'Parameters'** quicktab.

The first three options control what happens to the inheriting particle as it gets closer to the target particle. If **'Snap to Target'** is turned on, the inheriting particle will snap to the same position as the target particle once the distance between them is less than the value in the **'Snap Distance'** setting. However, if the velocity of the inheriting particle is not the same as that of the target particle, it may not remain snapped. See the example file [ch16_mods_inherit_snap.c4d](#). The blue particle chases the orange particle, overshoots a little (because its velocity is higher than the blue particle at that point - it needs to catch up), then slows down to fix the overshoot. By that time, the 60 frames to complete the operation have elapsed and the modifier makes no more changes to the parameter, with the result that the blue particle starts to fall behind. To fix, turn on the **'Keep Snapped Position'** switch. Now the blue particle still overshoots but quickly snaps to the target particle position and stays there.

⚠ There is a caveat to be aware of when the **'Keep Snapped Position'** switch is turned on. See the example file [file]. When you play it, the blue particle is noticeably offset from its target. Why is this? The target particle is emitted from a vertex of a Cube object and the **'Stick Particle to Source Object'** switch is turned on. The orange particle is therefore stationary but it still has a velocity (speed and direction) set internally. The modifier moves the inheriting particle to the target's position, and to do so it gives the blue particle the same velocity as the target particle - even though that is not actually moving. The result is that each frame the inheriting particle has the same velocity as the target particle so is moved with that speed and direction away from the stationary target - and the modifier then tries to snap it back again. The solution is fortunately simple: set the target particle speed to zero and the problem disappears.

The final options are the 'braking' settings, which are only available in one of the two distance inheritance modes. See the example file [ch16_mods_inherit_brake.c4d](#). Here, the inheriting (blue) particle moves to the target particle's position at a constant speed until it finally snaps into position. You might prefer that as the two particles became closer, the blue particle would slow down so that it eased into position more smoothly. Turn on the **'Use Braking'** switch and now the blue particle slows down when the distance between the two is less than 100 screen units.

You can control when braking starts with the **'Brake Distance'** setting and how strong the braking is with the **'Brake Strength'**. Generally speaking, the larger the distance the lower the strength should be, or the particle will slow too much. You can stop the particle slowing to zero (if desired) by setting the **'Min Speed'** value; the particle's speed will then never drop below this. You may also want to reduce the **'Snap Distance'** setting to avoid a sudden jerk at the end of travel. Try setting this to 1 or 2 screen units in the example file to see the difference.

16.9 Kill modifier

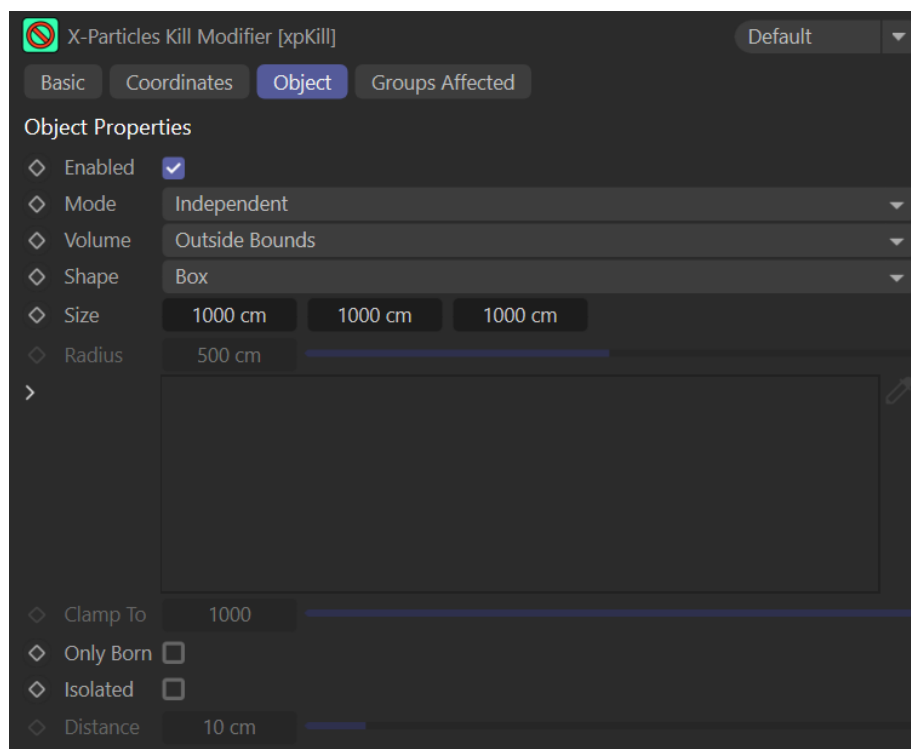


Figure 16.15. Kill modifier

X-Particles provides a number of ways to get rid of unwanted particles. You can use a Particle Life modifier, a Change Life action, or this modifier. The difference with this modifier, however, is that you can set an area outside of which (or inside of which) any particle will be removed from the scene.

What it does

All this modifier does is kill a particle when the specified criteria are met. With one exception, the modifier tests the position of a particle. You can specify whether they are killed if they enter an area, or if they leave an area, or if they are no longer present in the current camera's field of view. The exception is a limit on the total number of particles, and in that case the particle position is not relevant.

Why use it?

Sometimes it can be very desirable that stray particles leaving (or entering) a certain area are removed from the scene. Consider the example scene [ch16_mods_kill_box.c4d](#). In this, there is a FlowField object which alters the particle velocity. Eventually, particles leave the FlowField box at which point the FlowField no longer controls them. It might be nice to remove those particles and make what the FlowField is doing a lot clearer. To see this, enable the Kill modifier in the object manager and play the scene. The modifier uses a box the same size and position as the FlowField and is set to kill any particles outside the bounds of that box.

How to use it

The most important choice is the type of volume, set in the **'Volume'** menu. **'Outside Bounds'** and **'Inside Bounds'** use a volume defined by the modifier itself, visible as a green box or sphere in the viewport. **'Outside Bounds'** will kill any particles outside the volume; **'Inside Bounds'** will kill any particles inside the volume. With these options you can choose between a box or sphere shape for the volume and adjust the size as required.

Instead of the modifier-defined volume, you can use one or more mesh objects to define the volume. In the example file [ch16_mods_kill_objects.c4d](#) the setup is similar to the previous file, with a FlowField object and a Kill modifier. The modifier is set to use objects, in this case a cube and a sphere, which are dragged into the list of objects to use. You can set each object to kill particles outside or inside their volume. To do this, you use the icon in the object list as in Figure 16.16.

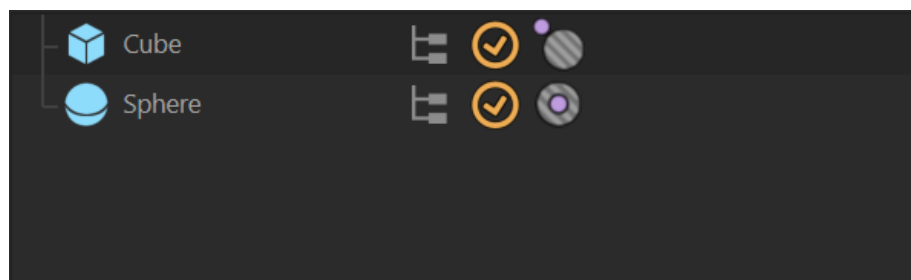


Figure 16.16. Kill modifier 'Objects' mode object list

The cube is set to kill particles outside its volume, but the sphere to kill particles which are inside its volume. Click the icon to change between the two states.

The next option is to kill particles which are outside the camera's field of view. This is much easier to use if you are using a camera other than the default, since its field of view is shown in the viewports.

The final option is '**Clamp to Max Particles**' which will cause the modifier to kill any particles which exceed the number of particles in the '**Clamp To**' setting.

There are two settings which are available whichever volume option is used. The first is '**Only Born**'. If this is turned on, the only particles which will be killed are those which have just been created; older particles are not affected. In the example file used above, [ch16_mods_kill_box.c4d](#), try turning on this switch. You will see that now no particles are killed, since only newly-created ones would be killed and no newly-created particles escape the box volume.

The other setting is '**Isolated**'. Regardless of the volume option, this will kill particles whose distance from every other particle is greater than the '**Distance**' setting. With the same example file (don't forget to turn off '**Only Born**') turn on this switch and set the distance to 5 units. When you run the scene, all particles are immediately killed if they are more than 5 units from all the other particles.

16.10 Particle Life modifier

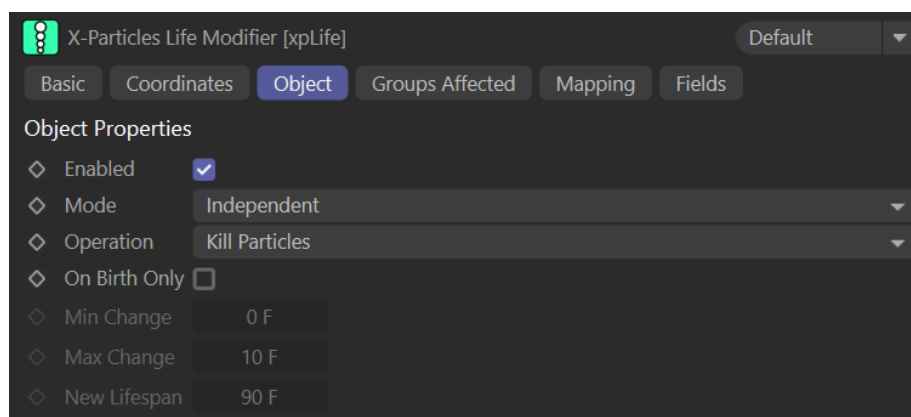


Figure 16.17. Life modifier

What it does

This modifier can be used to alter a particle's lifespan (note, the lifespan not the age - that remains unchanged). It has a number of ways to do this.

Why use it?

The principle use of this modifier is probably to kill particles. That is the default operation and means that any particle entering its

field of effect is immediately killed and removed from the scene. You could also use a Kill modifier to do this, which is generally a better option as it offers more control. However, the Kill modifier does not use fields (or falloffs, pre-R20), nor does it offer any data mapping.

Other than killing a particle, the modifier can change the lifespan without necessarily removing the particle. You might, for example, determine that when a particle enters a certain region its lifespan is to be reduced. It's not a very common requirement but it's there if you need it.

How to use it

First, select the operation required. **'Kill Particles'** is the default and there are no other parameters to set. Alternatively, you can increase or decrease the lifespan. To do this, set the **'Min Change'** and **'Max Change'** settings to the required values. Each frame, the lifespan will increase (or decrease) by a random time value between the minimum and maximum limits. However, this is also dependent on falloff. If a field (or falloff) is used, the actual change will be adjusted depending on where the particle is in relation to the falloff inner zone; inside the inner zone, the full change is applied. Be aware that the change occurs every frame while the particle is in the modifier's field of effect.

The remaining two options are to set a new lifespan. If you choose **'Set New Lifespan'** this will set the lifespan to the time in the **'New Lifespan'** setting. This happens regardless of particle age, so it is possible to set a new lifespan which is less than the particle's current age; in that case, the particle will immediately die.

The last choice is **'Set Lifespan Relative to Age'**. What this does is randomly select a value between the minimum and maximum limits, and adds that to the current age. You can set a specific change by setting the minimum and maximum values to be the same. Note that negative values are not possible. If you think about it, if the particle age was 30 frames and the value to add was -1 frame, the lifespan would then be 29 frames. Since this is less than the age of 30 frames, the particle would immediately die, and with this option that will always be the case - so negative values are pointless.

There is one other option - **'On Birth Only'**. It can only be used if the operation is set to kill particles, but if it is turned on only newly-created particles will be killed. This might seem odd because the switch makes no difference if particles are always in the modifier's area of effect. If the modifier uses a field (or falloff), however, it makes quite a difference. See the example file [ch16_mods_life.c4d](#). This uses a spherical emitter with a Life modifier which has a spherical field overlapping the emitter. If you play the scene, when the particles are first created those within the field are instantly killed. As they start to move, any which then enter the field are also killed, so the eventual result (since the field is a little smaller than the emitter) is to produce a ring of particles like this:

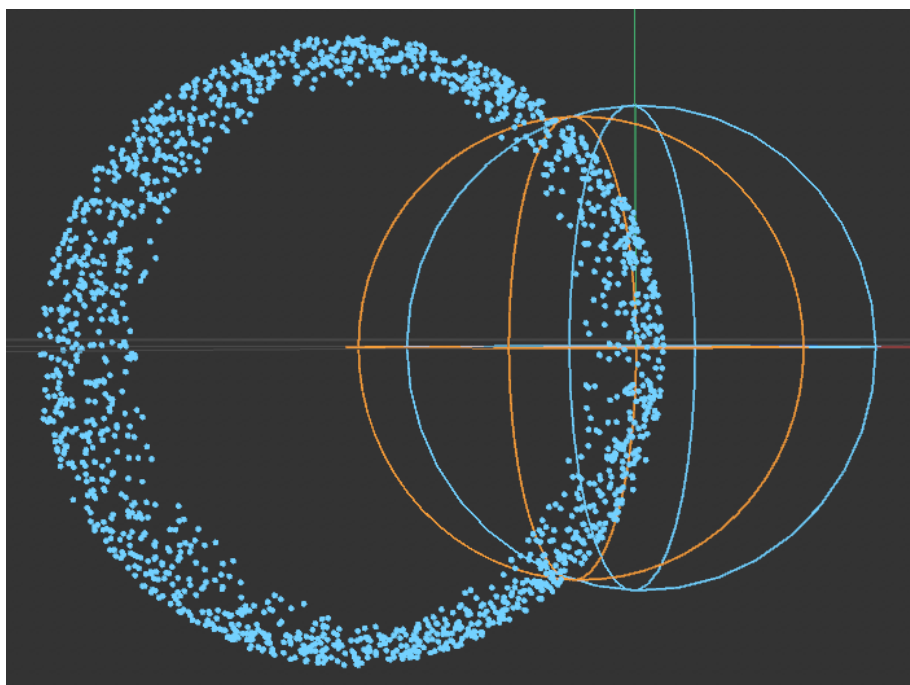


Figure 16.18. Possible use of the 'On Birth Only' switch

Now turn on the **'On Birth Only'** switch. With this on, only newly-created particles are killed and any which subsequently enter the field are not. This gives a sphere of particles with a smaller sphere removed from the centre (you need to try the scene to see this, it

doesn't show up well on a screenshot).

16.11 Negate modifier

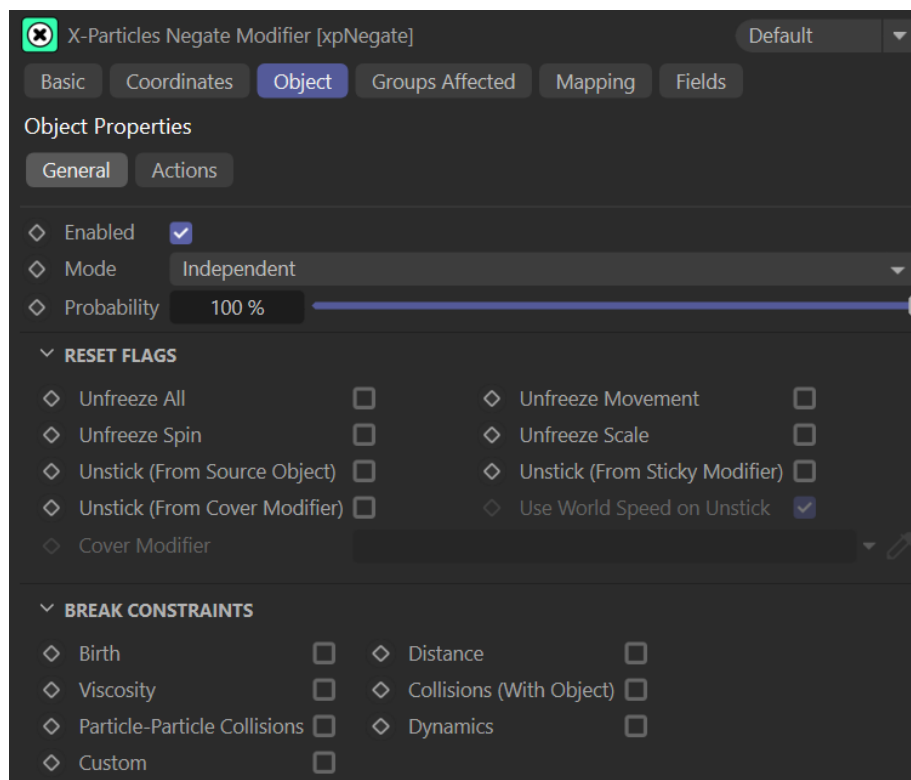


Figure 16.19. Negate modifier

What it does

The Negate modifier resets particle flags or breaks previously-formed constraints.

Flags have been mentioned several times in previous chapters. They are signals set in the particle which instruct some other object in XP to do something (or not to do something) or to inform an object that the particle is in some kind of state the object should be aware of. For example, the particle freeze flags instruct the emitter not to move, or spin or scale, a particle while the flag remains set. Another frequently-used flag tells a Cover modifier that the particle has arrived at its target destination on an object and also tells the emitter that the particle must only be moved if the object it is stuck to moves.

Clearly, it's useful to be able to clear these flags if a frozen particle needs to start moving again, or to release a particle stuck to an object. There are several ways to do this. To unfreeze a particle, for example, you could use:

- a Freeze modifier in unfreeze mode
- or a Freeze action, again set to unfreeze
- or a Negate modifier

The Negate modifier was added initially to break constraints formed between particles and other particles or scene objects, and was expanded to allow clearance of a number of flags. Note that the modifier cannot set any of these flags, it only clears them; likewise, it cannot make constraints, only break them.

Why use it?

The clearing of flags has been discussed elsewhere and there is no need to repeat it here. See Chapter 13, section 13.5.5 'Unstick from source'; section 13.8.2 'Actions with other functions than altering particle data'; and this chapter, section 15.2.3 'Freeze modifier'.

Constraints haven't been covered so far in this book, but they are connections formed between particles or between particles and scene objects in response to a number of conditions. The Negate modifier can break these constraints.

See the example file [ch16_mods_negate_constraints.c4d](#). This uses a Collider tag to make a constraint between a particle and the plane object when the particle collides with the plane. This is shown as a yellow line between the particle and the object. The Negate modifier is animated and when particles enter its field of effect the constraints are broken. The action is there simply to show when the constraint is broken; when that happens, the particle turns red but this is optional and used here only to demonstrate what is happening. If you watch the animation carefully, you can see that all the constraints are initially broken but then, depending on the particle's movement once it is released, it may collide again with the plane and the constraint will be reformed. This makes an important point about this modifier: it does not prevent constraints from being made, it only breaks those which already exist, and they can be made again.

How to use it

There aren't many options in this modifier. All you need to do is select which flags to clear and/or which constraints you want to break. You can also add one or more actions which will be triggered when the modifier clears a flag or breaks a constraint. The only other setting is **'Probability'**. This gives you some more control over what the modifier does. If the setting is at the default of 100%, the modifier will always clear the flag or break the constraint. If it is less than that, it may not do so, but as long as the particle remains in the field of effect that will be retested each frame. In the example file, try reducing this to 5% and you will see some of the constraints are never broken. You can tell which constraints are not broken if the particle remains blue. A red particle with a constraint is one where the constraint was broken, so the action was triggered to change the colour, but was then remade.

16.12 Physical modifier

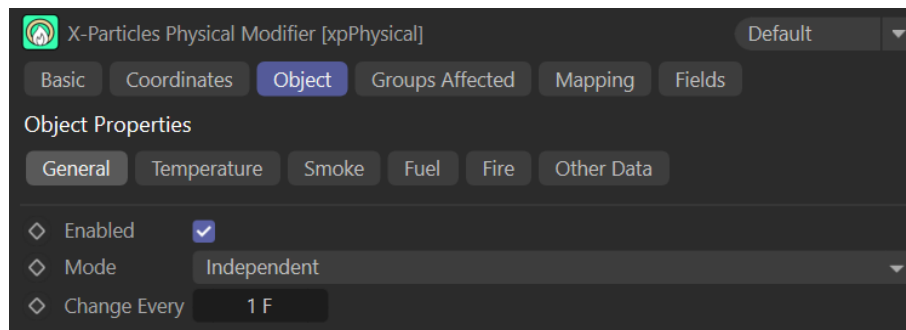


Figure 16.20. Physical modifier

At first sight, this looks like a large and complex modifier, but in fact all it does is change certain 'physical' parameters used in simulations of fire, smoke, liquid, etc.

What it does

The modifier has separate tabs to control the change in temperature, smoke, fuel and fire, plus another tab with three more assorted parameters. In each case you can change the value of the parameter over time, or simply set a new value. In all tabs except the **'Other Data'** tab you can also use a shader to change the parameters.

Why use it?

During a simulation you might want to change one or more of the parameters used. See the example file [ch16_mods_physical_friction_1.c4d](#). This is a very simple fluid simulation. If you play it, you see that the amount of friction between the particles and the Plane object is very small – the particles flow over the surface easily and fall off the sides of the Plane. Now enable the physical modifier in the object manager and play the scene again. The modifier is increasing the friction by a small amount each frame until it becomes very high and movement is almost halted.

You can also use the modifier to reduce friction if required. See the example file [ch16_mods_physical_friction_2.c4d](#). Initially the friction value is very high so when the particles hit the inclined Plane they almost stick where they are. But the modifier is set to reduce the friction, and as it lessens, the particles start to slide faster and faster down the Plane.

i Each particle is given its own internal friction value by the emitter when it is created; this is set in the emitter's Extended Data tab, Physical Data quicktab. The collider tag in this scene also has a friction value, which you can regard as the friction of the surface, and the final friction used by the simulation is calculated by multiplying the particle and tag values together. The physical modifier increases the particle friction value before the multiplication occurs. The implication of this is that if the particle friction is (for

example) 80%, but the tag value is zero, the final friction result will always be zero - and the modifier has no effect at all. This may occur with other parameters too, so be aware of this when using this modifier.

How to use it

In the General tab, there is only one setting other than those common to most other modifiers. That is how often the change in the physical parameter occurs. The default is for the parameter to be updated each frame, but this can be slowed down by increasing the **'Change Every'** setting.

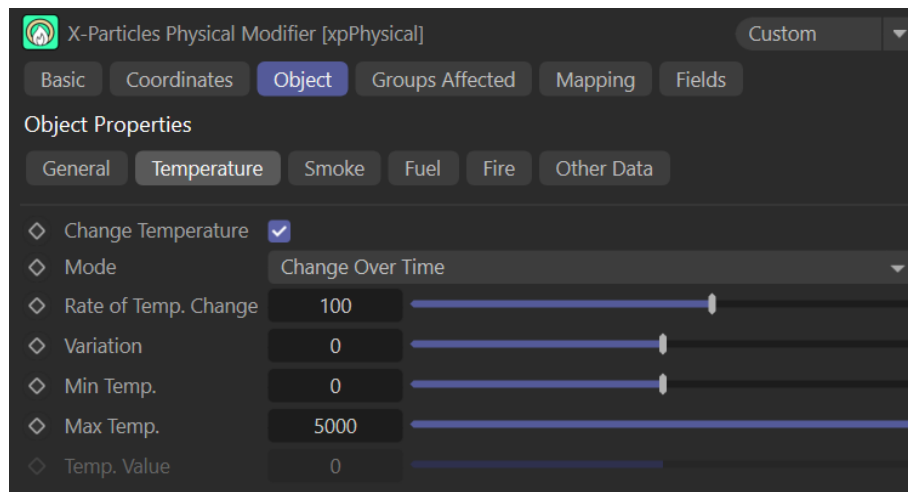


Figure 16.21. Physical modifier 'Temperature' quicktab

The Temperature, Smoke, Fuel and Fire tabs are identical in use apart from their default settings (see Figure 16.21). Taking temperature as an example, you must first turn on the switch **'Change Temperature'**. Similar switches in the other tabs let you change any or all of the parameters, or change some without affecting others. Then you select the mode to use. The simplest is **'Set Value'** which immediately sets the particle temperature to the value in **'Temp. Value'**. The next mode is **'Change Over Time'** which changes the temperature by the amount in **'Rate of Temp. Change'**. You can apply some variation to the rate of change between different particles by using the **'Variation'** setting. How often this change occurs depends on the **'Change Every'** time setting discussed above. Note that the rate of change can be negative, which will reduce the temperature rather than increasing it.

The **'Min Temp.'** and **'Max Temp.'** settings apply limits so the values don't get too extreme. The modifier will not change the particle temperature to a value outside either of these limits.

The final mode is **'Use Shader'** and in this mode you can use a channel shader to drive the temperature. See the example file [ch16_mods_physical_shader.c4d](#). In this scene, the particle display (in the emitter) is set to **'Gradient (Parameter)'** with the parameter being the temperature. When the shader is sampled the overall colour brightness is calculated and that is used to select a temperature value between **'Min Temp.'** and **'Max Temp.'**. The emitter then assigns a colour to the particle from the gradient, the colour selected depending on the particle temperature. In this scene the shader is a Fire shader, and since that is animated the particle colours change as the scene runs.

When you use this mode, a number of additional settings become available. These are exactly the same as those discussed in the Color modifier when you use a shader there. For details, please see this chapter, section 16.3, 'Color Modifier'.

The other tabs all work in the same way, with the same settings except that you cannot change friction, bounce, or the user value with a shader.

i The **'User Value'** has no meaning for X-Particles. It is a floating-point value you can set to anything you like; the physical modifier can change it and the Question object can test its value but that's all it does. It is exactly the same as adding an item of custom data to the particle (see Chapter 5, section 5.5, 'Custom Data' for details) but is less flexible than custom data since it can only be a single floating-point number.

i You might be wondering why particle mass, which is set in the emitter's Extended Data tab, Physical Data quicktab, is not altered by the Physical modifier. This is for historical reasons. Mass, as a particle parameter, existed long before temperature, smoke, etc. so there was no Physical modifier at that time. To change mass the Scale modifier was used, and to avoid breaking existing scenes changing mass was kept there when the Physical modifier was introduced.

16.13 Python modifier

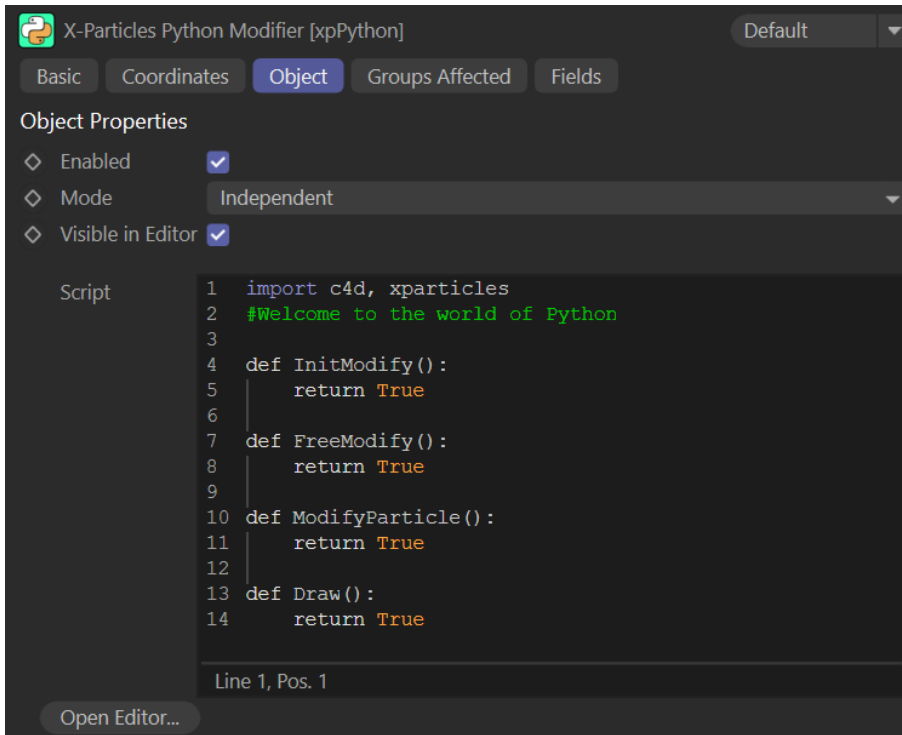


Figure 16.22. Python modifier

What it does

The Python modifier executes a Python script on particles each frame. You can use it to affect all particles or only those which meet certain criteria. It can be used to affect only those particles in its field of effect or any particle.

Why use it

This is a convenient way to change a particle parameter under complete control, or to change several parameters at the same time which would otherwise multiple modifiers to do so. What it can do is really only limited by your imagination.

How to use it

This modifier works like any other, including the use of fields (or falloff pre-R20), particle groups, and whether it works independently or is action-controlled. You only have to provide the Python script to execute. See the example file [ch16_mods_python_1.c4d](#). This executes the following script:

```
import c4d, xparticles
#welcome to the world of python
def ModifyParticle():
    tm = particle.GetTime()
    if tm > 0.5 and particle.GetRadius() < 10:
        particle.SetRadius(particle.GetRadius() + 0.2)
        particle.SetColor(particle.GetColor() + c4d.Vector(-0.03, 0, 0.03))
    return True
def Draw():
    return True
```

Each frame, the modifier iterates through all particles in its field of effect and does the following:

- gets the particle age
- tests if the age is more than 0.5 seconds and the radius is less than 10 scene units
- if so, increases the radius by 0.2 scene units and changes the colour

The final result is particles which have a radius of 10 units and are coloured blue. You can see that to do this without python would require a Scale and a Color modifier, plus a Question to activate these modifiers when the age was greater than 0.5 seconds. The example file [cb16_mods_python_2.c4d](#) shows the same effect without using Python; I think you'll agree that the Python version is more concise and requires one object in the scene rather than three.

More details on using Python in X-Particles will be covered in a later section in this book.

16.14 Scale modifier

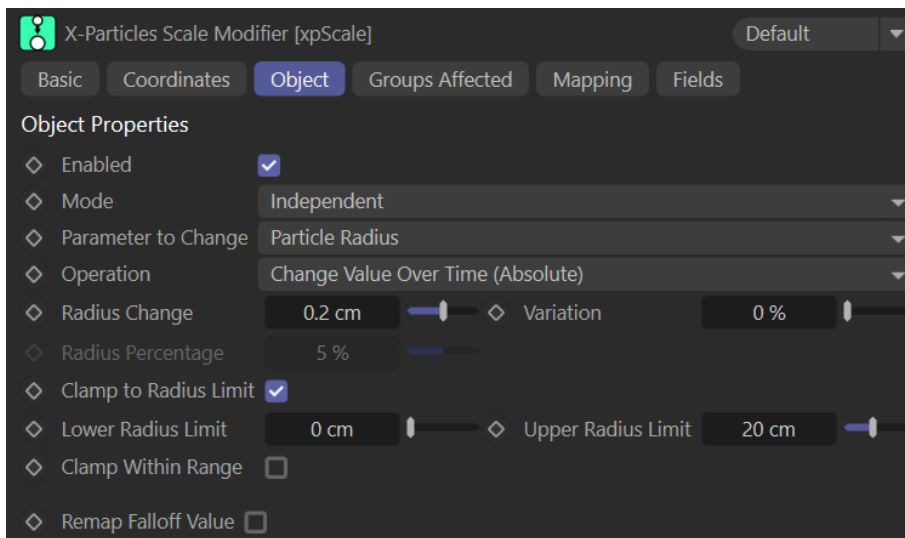


Figure 16.23. Scale modifier

What it does

The Scale modifier changes one of three particle parameters, scale, radius or mass. The original version changed the scale (hence the modifier's name) but when it became clear that radius was a much more useful parameter, this became the default, but the name didn't change. Mass was added as a third option before the Physical modifier existed, but stayed in the Scale modifier to avoid older scenes breaking.

Hopefully that makes it all clear. Hopefully.

Why use it?

Put simply, you can use it to change one of these three particle parameters; it does nothing more than that.

How to use it

The first thing is to decide which parameter you want to change. Selecting radius or mass gives you the same interface except that some of the setting names change (they either read 'radius' or 'mass') and the defaults are changed to be appropriate for each one. With scale, the interface looks different but that's mostly because scale is a three-value vector field and the two-column interface used for radius and mass doesn't work so well. There is also an extra parameter with scale - '**Jiggle Variation**'. Other than that, the chosen parameter doesn't affect the interface, so while this section will only use radius for any examples, you can assume that the same applies to the other parameters unless otherwise stated.

Once you have chosen the parameter to change, you must select the '**Operation**', which is simply the method used to change the parameter. There are a number of these, and they work as follows.

1. Change Value Over Time (Absolute)

This is very simple: each frame, the modifier will add the value in the **'Radius Change'** (or **'Mass Change'** or **'Scale Change'**) setting to the particle radius (or mass or scale). If you want to reduce the radius, use a negative value for **'Radius Change'**. See the example file [ch16_mods_scale_abs_1.c4d](#). If you want to stop the radius from going outside limits that you decide, turn on the **'Clamp to Radius Limit'** and set the desired upper and lower radius limits in the respective fields. In the example file, try turning the clamp off and see the difference. Finally, the radius change will be the same for all particles unless you set the **'Variation'** field to greater than zero, in which case the change will be different for each particle. This is shown in the example file [ch16_mods_scale_abs_2.c4d](#).

2. Change Value Over Time (Relative)

This works in exactly the same way as the absolute method, but instead of specifying a radius change value in scene units, you specify a percentage change, which again can be negative. But a percentage of what? In this case, the value added to (or subtracted from) the particle radius depends on whether the **'Clamp to Radius Limit'** switch is turned on. If it is, the value used is the difference between the upper and lower radius limits multiplied by the **'Radius Percentage'**. As an example, suppose the difference between the limit values is 10 scene units and the percentage value is 5%. In that case, the modifier would add 0.5 scene units to the radius each frame (because $10 \times 0.05 = 0.5$) and this will be the case no matter what the initial value of the particle radius was.

However, if the clamp switch is off, the value added to the radius each frame is the initial radius value multiplied by the percentage value. Using the same example, if the initial particle radius was 1 unit, each frame the modifier would add 0.05 units to the radius (because $1.0 \times 0.05 = 0.05$); but if the initial radius was 5 units, the modifier would add 0.25 units to the radius each frame ($5.0 \times 0.05 = 0.25$).

3. Set Value

The simplest mode possible: the radius is set immediately to the value in the **'Radius Value'** field.

4. Set by Falloff

In this mode, there is no radius value to set. Recall that a field (or falloff, pre-R20) always returns a value between zero and one. The initial value of the particle's radius is multiplied by the inverted falloff value (that is, an actual falloff value of zero becomes 1, while a value of 1 becomes zero) and the particle radius is set to the resulting value. See the example file [ch16_mods_scale_falloff_1.c4d](#). The particles have an initial radius of 10 units. The falloff value outside the field object is zero, but this is inverted and becomes 1 - so the radius is unchanged outside the field. As the particles enter the field, the falloff value starts to increase - but it is inverted, so in the inner falloff zone the inverted value is zero. In this scene, the lower radius limit is clamped to 2 units, but if that wasn't the case the particles would disappear in the centre of the field because their radius would be zero. But then, as the particles move out of the falloff, the inverted falloff value increases back to 1, and the particles regain their original radius.

What if you wanted to use this method to increase the radius instead of decreasing it? This needs a bit more thought. First, you would need to invert the inverted falloff value using the switch in the field/falloff object so that it returns zero outside the falloff zone and one in the inner falloff zone. That would mean that the particle radius outside the zone was zero, but we can fix this by setting the lower radius limit to the desired value. For the result, see the example file [ch16_mods_scale_falloff_2.c4d](#).

5. Jiggle

This does exactly what the name implies. Each frame a value between zero and the **'Radius Change'** setting is either added to or subtracted from the particle radius. This makes the radius jiggle around its initial value but always remains around that value. For particle radius and mass you can add some variation to the jiggle amount with the **'Variation'** setting. For scale this is not available but you can add variation with the **'Jiggle Variation'** setting which is a single value applied to all three components of the scale vector.

6. Use Spline

With this mode, the actual value of the radius (or scale or mass) is obtained from the **'Scale Spline'**. The horizontal axis of the spline represents the scene length while the vertical axis is the value of the parameter. By default, the maximum value of the scale spline is 2, because this was originally intended to work with particle scale. For radius, you may want to increase the **'Max Scale'** value so that larger radius values are usable. The example file [ch16_mods_scale_spline.c4d](#) shows how the spline works, with radius values pulsing between 1 and 10 scene units over the length of the scene. Try switching to particle scale instead and you will see why the maximum scale value was restricted to 2!

7. Scale by Speed

This mode is similar to **'Set Value'** in that the radius is not changed each frame, but is set using the particle speed and the value in the **'Radius Change'** setting. For example, if the speed is 150 units and the radius change value is 0.03 units, the radius will be set to 4.5 units ($150 \times 0.03 = 4.5$). The radius will therefore only change if the particle speed changes (or of course if you keyframe the radius change setting). In the example file *ch16_mods_scale_speed.c4d*, the particles all start with the same speed and radius, but a speed modifier increases the speed each frame, the increase varying between particles. The result is that the faster the particle is, the larger it becomes.


8. Scale by Acceleration

This is similar to 'Scale by Speed' but in this case the particle radius is changed each frame depending on the change in particle speed. If there is no change in speed, there will be no change in radius. In the example file *ch16_mods_scale_accel.c4d* the particles have the same speed and radius when created; if you play it, you can see that the radius doesn't change because the speed does not change. Now enable the Drag modifier in the object manager. This will slow the particles down and you can see that as they slow the radius is reduced.

9. Use Shader

This final mode uses a channel shader to alter the radius. The actual radius value is calculated from the setting of **'Radius Value'** multiplied by the overall brightness returned by the shader at the point it is sampled, which is obtained from the particle position. The example file *ch16_mods_scale_shader.c4d* makes this clearer. Here, a Noise shader is used in the modifier. This is a 3D shader so as the particle moves through the 3D world, the colour returned from the shader varies and the radius is changed. You can use 2D shaders as well (e.g. Checkerboard or a bitmap) but these won't return different colours as the particle moves, unless the shader is animated.

There are several other controls which let you adjust the scale (of the shader, not the particle!), offset and tiling, and to mirror the shader if desired. These are exactly the same as those discussed in the Color modifier when you use a shader there. For details, please see this chapter, section 16.3, 'Color Modifier'.

 Note that this shader is marked as deprecated in favour of data mapping so there is a possibility that it will be removed in a future version of X-Particles.

16.15 Sound modifier

The basic interface of this modifier is shown in Figure 16.24.

What it does

The Sound modifier is interesting because, like many modifiers, it can change particle parameters such as colour, radius, speed and so on. However, it can also change the rate of emission of particles from an emitter; in both cases, it uses a sound file to drive the changes being made.

The modifier looks complex at first glance but it's actually quite easy to use when you understand a little about how it works.

What the modifier does is to sample a sound file while the scene plays. It samples the amplitude of the sound (i.e. the strength of the sound) and the range of frequencies used; it also measures the strength of each frequency. The results are used to drive changes such as selecting a colour from a gradient or setting the radius from a range you specify.

Why use it?

This is very much one of those areas which require some creativity and imagination. But you could - for example - use it to drive particle emission in time to music, which looks nice. There are a lot of other potential uses but you will need to do a lot of experimentation to find the best settings for whatever project you are hoping to develop.

How to use it

The first thing you will need is a sound file. Without that, the modifier can do nothing. You can use any sound you like, but the file must be in WAV format. R25 and up have a file called *technoloop_2_kick.wav* in the asset browser, which can be used but isn't very

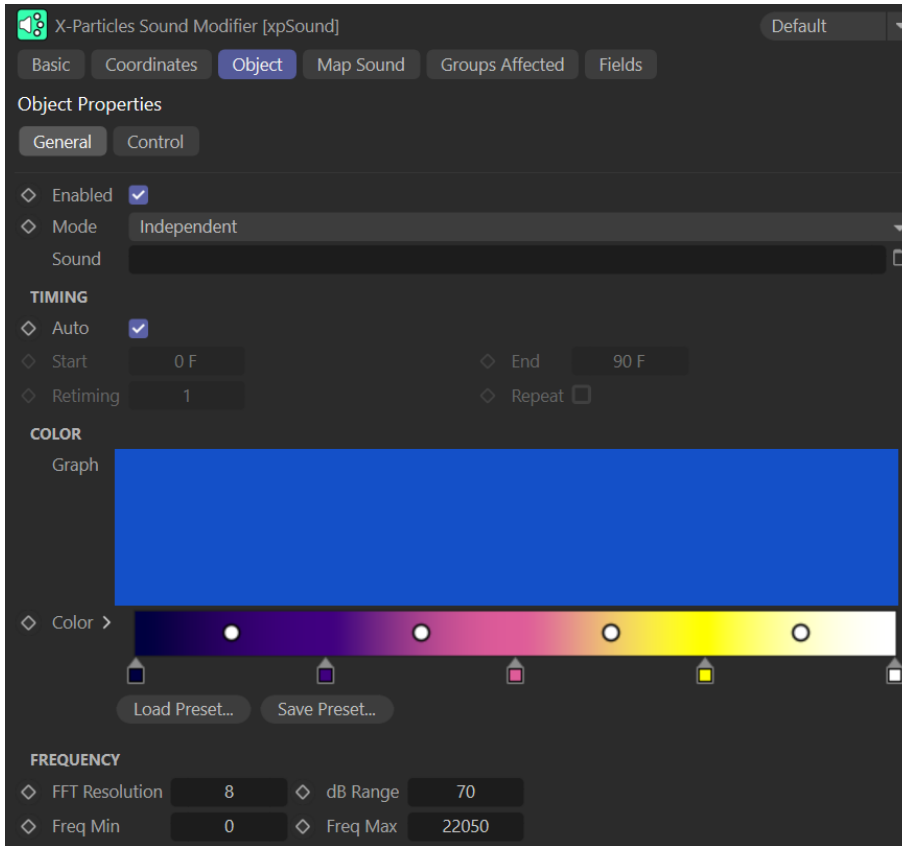


Figure 16.24. Sound modifier

exciting. For the purpose of this section I will be using a file downloaded from the website Noiiz (<https://www.noiiiz.com>) named 120_FullBeatLoops_02_2_SP_449.wav. I can't distribute this but you can register for free at that site and download the file if you wish. It's a drum loop which works quite well to demonstrate the effect of this modifier.

⚠ One limitation of the Sound modifier is that it can't actually play the sound. To play it, you will have to add the sound as a special track in the timeline. The Cinema 4D documentation tells you how to do this but it isn't that clear, so if you need it, the method is shown at the end of this section - see [link to method].

1. Sound graph

A reference file for use with this section is named *ch16_sound_basic.c4d* and is in the download archive for this chapter. All you need to is supply the .wav file to use, which you add to the 'Sound' link field in the modifier. If you do that, then with the sound file I'm using you will see this in the modifier:

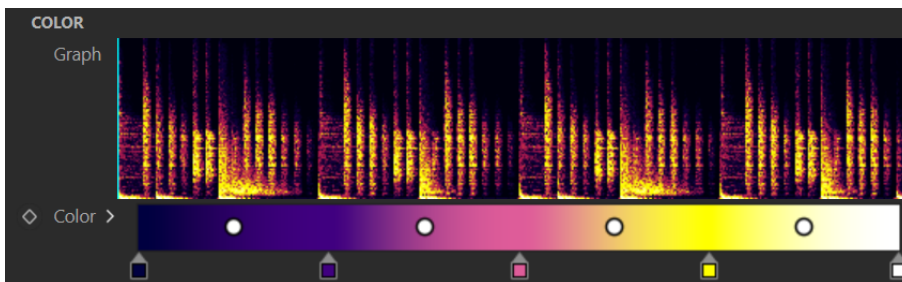


Figure 16.25. Sound graph with loaded sound file

This graph shows the distribution of frequencies in the sound along the length of the clip. Each vertical line is one sample from the clip, with the low frequency at the bottom of the Y-axis and high frequency at the top. The strength of each frequency in each sample is shown by a colour from the 'Color' gradient, so with this gradient the strongest frequencies are yellow to white, while the least strong are purple to dark blue. You can see from the graph that this sound is predominantly low to medium frequency with very little high frequency sound, which is what you might expect from a drum loop.

The frequency range can be altered using the **'Freq Min'** and **'Freq Max'** settings in the frequency section below the graph. This acts as a filter to remove sound outside those limits, but much of the time you can leave these alone. Note that the colour gradient is only used to display the graph and has no effect on the particle colours.

In the reference file I have changed the emission rate and particle display type, plus turned off subframe emission, to make it easier to see what is happening. If you play the scene, nothing happens and that is because to do anything the sound needs to be mapped to one or more particle parameters (or to the emitter emission rate). The mapping process simply takes the peak frequency or amplitude of the sound at each point during the animation and converts that into a colour or a particle parameter using a colour gradient (for colour) or a value range that you supply for other parameters.

To add a map is very simple, just click the **'Add'** button in the **'Map Sound'** tab, then select the parameter to map from the **'Param'** menu. There are a range of parameters to choose from. Most are the usual ones such as colour, radius, speed etc. but you can also change any custom data you may have added, or instead of affecting the particles you can use the sound to change the emission rate from the emitter. You can delete an existing map with the **'Remove'** button in the map or just turn off the **'Enabled'** switch to disable the map without removing it.

2. Colour mapping

The file contains two maps which have already been added, but both are disabled. Go the **'Map Sound'** tab and enable the first map (Color) so the interface looks like this:

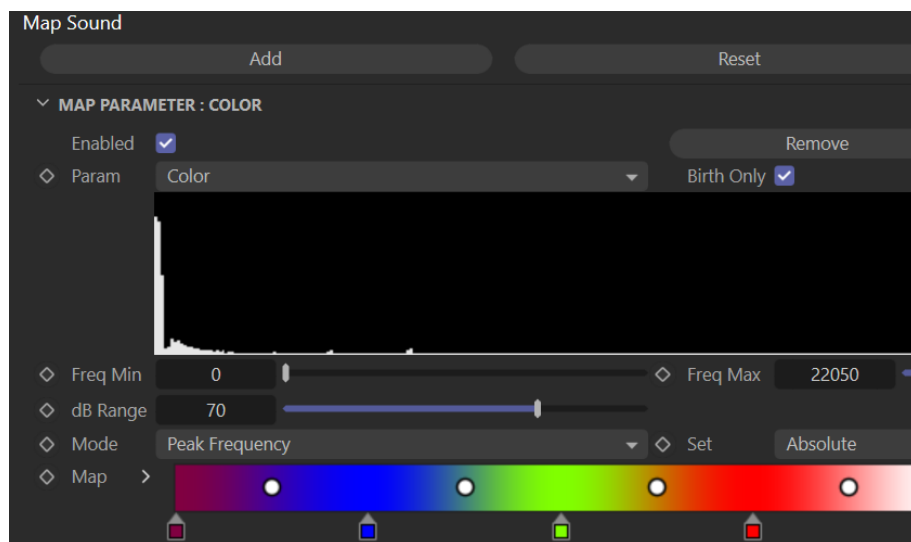


Figure 16.26. Mapping sound to particle colour

You can see that this will map the sound to particle colour - in other words, the colour will change depending on the sound. All the settings are the default ones, except **'Mode'** which is set to **'Peak Frequency'**. There is another graph here and for the current frame this shows the frequency distribution (along the horizontal axis) and the amplitude (strength) of each frequency along the vertical axis. This graph changes each frame because the frequency distribution will change as the sound plays.

Now enable the colour map by turning on the **'Enabled'** switch (leave the radius map disabled for now). On playing the scene, you see this:

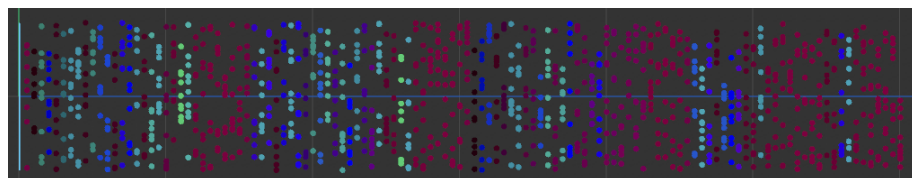


Figure 16.27. Result of colour mapping

Each frame, the modifier determines the peak frequency in the sound at that frame and maps that to a colour in the **'Map'** gradient. As you can see, most peak frequencies are coloured magenta to blue, indicating that these are low frequencies, which was already expected from this file. To get a more even spread of colours we can set the **'Freq Max'** value to a lower number; try 5,000 in this case and then you see this:

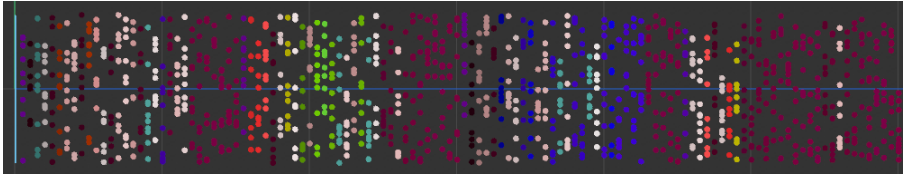


Figure 16.28. Colour mapping with low 'Freq Max' setting

What this does is exclude the higher frequencies and uses the full gradient for the remaining ones, which are mostly low ones. Note also that the graph for this map changes - it becomes more blocky, because the range of frequencies has narrowed considerably. Next, if you revert to the default '**Freq Max**' of 22,050 and now set '**Freq Min**' to 10,000 this is the result:

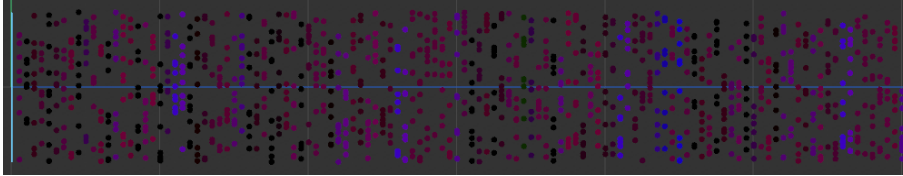


Figure 16.29. Colour mapping with high 'Freq Min' setting

The particle colours are darker now because we are excluding the low frequencies which predominate in this clip and the peak frequencies which remain are small because this file doesn't have strong high frequency sounds.

Instead of peak frequency you can use peak amplitude instead. This doesn't care about frequencies, only the maximum amplitude of the sound each frame. This is a loud file so as you would expect, the result is that most particles have colours from the right hand end of the gradient. In the same way as the frequencies, you can alter the amplitude range with the '**dB Range**' setting. If you increase it to 100, the colours are shifted even more to the right of the gradient, but if you reduce it to 20, the high amplitude samples are excluded and the colours become darker. In fact, many particles will be black, indicating that the peak amplitude in the that frame was higher than 20.

The colour map has a third mode, '**Composite**', which is a combination of peak frequency and amplitude. This is the only map type to use this mode, all the others only have peak frequency or peak amplitude.

There are two other controls to mention for all maps, with one or two exceptions. The first is '**Birth Only**'. If this switch is turned on the modifier will only affect newly-created particles once and not after that. Often this is desirable - you probably don't want to constantly change particle colour each frame as it will look like a really bad flicker. On the other hand, if you were mapping sound to particle speed you might want to update the speed each frame rather than change it just once. Use of this switch depends on the effect you want to achieve.

The other control is the '**Set**' menu. This controls how the parameter value is used. There are four options:

- Absolute: the particle parameter is simply changed to the calculated value
- Add: the calculated value is added to the current parameter value and the parameter changed to the result
- Multiply: the calculated value and the current value are multiplied together
- Subtract: the calculated value is subtracted from the current value

3. Radius mapping

Other parameters are mapped in the same way as colour but there are some additional settings. Load the reference file again and this time enable the second map but leave the colour map disabled. The second map is mapping sound to particle radius and is shown in Figure 16.30.

This looks similar to the colour map, except that there is no colour gradient and instead there are two limit settings labelled '**Min**' and '**Max**'. In addition, there is a spline control labelled '**Map**'.

The limit settings are the limits the modifier will place on the calculated value, which will then fall somewhere between those limits. It is up to you to make sure these are sensible values for your scene. For example, if mapping particle radius, you might choose limits of 1 to 10 units, but this would probably be inadequate for speed - you might map that to between 50 and 300 units, for example.

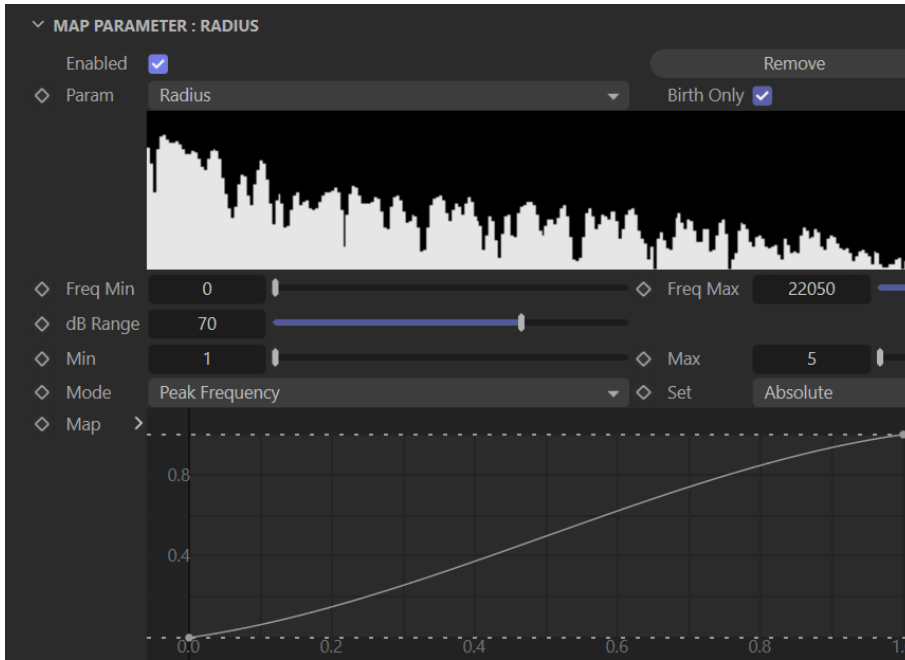


Figure 16.30. Mapping sound to particle radius

i The calculated value will not exceed the limits in **'Min'** and **'Max'** so that, if the calculated value is lower than **'Min'** the final value used will be the **'Min'** value. Similarly, the **'Max'** value will never be exceeded even if the calculated value is higher than that. However, this clamping is done before the **'Set'** menu is considered, so if that is set to **'Add'** it is possible that the final value might be higher than the **'Max'** setting.

The **'Map'** spline governs how the sound is mapped to the parameter. The horizontal axis is the peak frequency (or peak amplitude) and the vertical axis is a value from 0 to 1.0 which is used to select a value between **'Min'** and **'Max'**. You can see from this file that the spline will assign small radius values to low frequency sounds and large values to higher frequency ones. Most particles are small, because the sound is predominantly low frequency so the mapped radius is small. You can change the spline to emphasise certain peak frequencies. For example, this spline would give much more weight to low-frequency sounds:

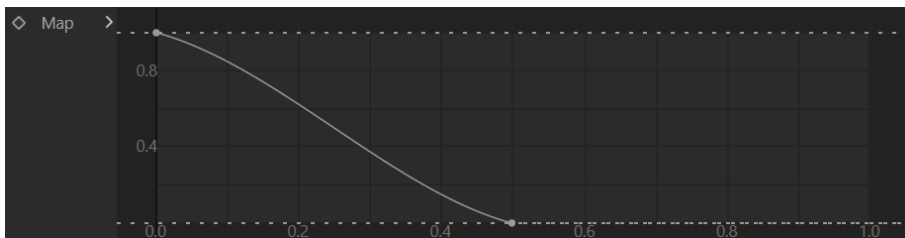


Figure 16.31. Map spline giving weight to low-frequency sounds

Figure 16.32 shows the result, comparing the default spline (top) with the edited spline (bottom).

4. Other particle parameters

These are self-explanatory for the most part. If you choose the **'Velocity.x'** parameter the sound will alter the velocity of the particle along that axis. The same applies if you choose the **'y'** or **'z'** components. This is not the same as mapping speed, which simply changes the speed regardless of direction. Changing the velocity may change direction as well as speed.

The only other setting in the maps is when you select **'Custom'** as the mapping parameter. Here you must enter the name of the custom data item to map; you set this in the emitter when adding the custom data item (see Chapter 5, section 5.5 'Custom Data').

5. Mapping emission

Instead of changing particle parameters, the Sound modifier can change the emission rate in the emitter. To do this, change the **'Param'** setting in the map to **'Emission'**. The other controls remain the same. The modifier will then change the number of particles emitted each frame by using the peak frequency or amplitude.

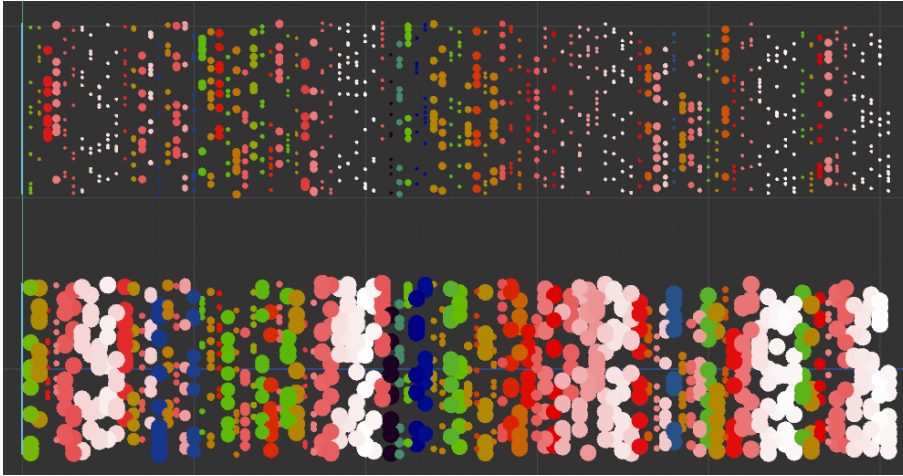


Figure 16.32. Effect of Map spline emphasising low-frequency sounds (bottom) on particle radius to the default Map spline (top)

But how many particles will be emitted? This depends on:

- the **'Birthrate'** setting in the emitter
- the **'Min'** and **'Max'** values in the map
- the **'Map'** spline
- and the frequency or decibel range which is set

The modifier first selects a value between the two frequency limits (or between zero and the **'dB Range'** setting) then uses that to find a value between **'Min'** and **'Max'**. This may be altered further by the **'Map'** spline and the final value is multiplied with the birthrate per frame.

For example, if the **'Birthrate'** setting in the emitter is 300, and the frame rate is 30 fps, then 10 particles would normally be emitted each frame. If the map has **'Min'** and **'Max'** values of 1 and 10 respectively, and the final value is 7, then $7 \times 10 = 70$ particles would be emitted that frame. This will vary each frame to give this sort of pattern:

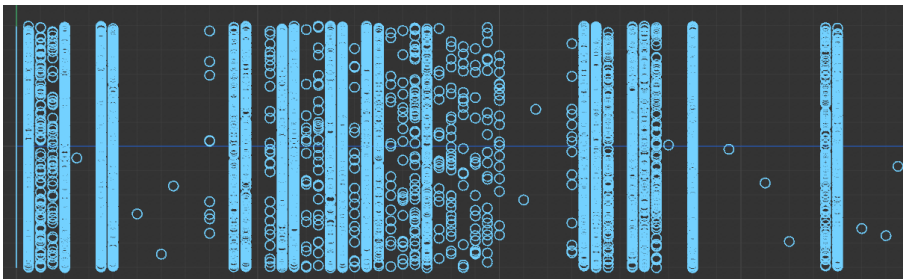


Figure 16.33. Mapping sound to particle emission rate

Note that subframe emission has been turned off for clarity.

6. Other Object tab settings

There are more settings in the Object tab that have not been discussed yet. The first section concerns timing. Most of the time, you can leave this set to **'Auto'**. However, if you turn that switch off, you can do several things:

- You can set the times when the sound clip starts and stops; if it stops before the scene ends, the modifier will return zero for peak frequency and amplitude and the particles will turn black (for colour maps), have a radius of zero (for radius maps) and so on
- Turning on the **'Repeat'** switch will loop the sound clip indefinitely
- The playback speed can be increased (by increasing the **'Retiming'** value) or slowed by reducing the retiming. Note that this does not affect the length of the playback; if that is set to 30 frames, for example, it will play for 30 frames regardless of the retiming value

The frequency section controls how the sound file is sampled. It does so with Fast Fourier Transform (FFT) a discussion of which is outside the scope of this book. The **'dB Range'**, **'Freq Min'** and **'Freq Max'** settings do the same as in the individual maps and

changing them will be reflected in the **'Graph'** display. The **'FFT Resolution'** controls the sampling resolution of the file. A lower value than the default 8 makes the sampling faster but resolution is lost in the process, leading to less detailed results. Higher values increase the resolution but take longer to process; be careful when increasing this setting or updating the graph may take some time and it may take a lot longer to play the animation.

7. The Control quicktab

This tab contains two splines which act as filters for amplitude (i.e. volume) and frequency. You can use these to remap the sampled values from the sound clip. For example, in the reference file with peak amplitude mapped to colour, then with default values we see the particles at the top, while below the modifier is the same except that the **'Amplitude'** spline has been flipped horizontally:

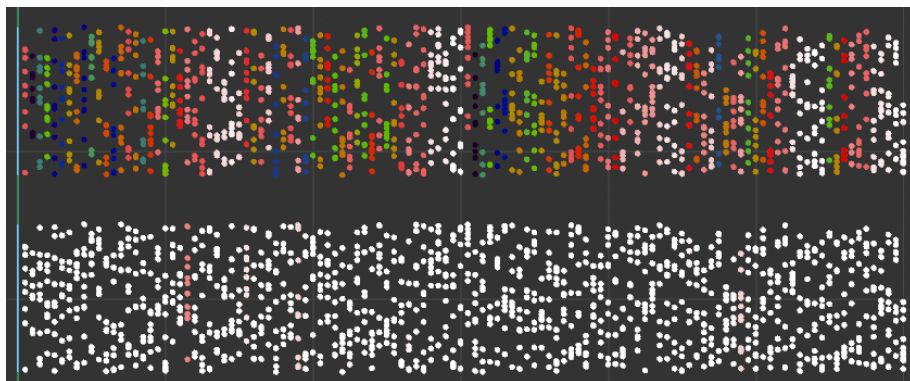


Figure 16.34. Effect of amplitude spline: default spline (top), horizontally-flipped spline (bottom)

At first sight this looks counter-intuitive. What it has done though is to remap the low-volume sounds to give a high result and the high-volume sounds to give a low one. The low-volume sounds now predominate and are all similar, giving a high peak amplitude and a white colour. To see more colours you would have to increase the **'dB Range'** setting in the map; if you increase that to the maximum of 100 you start to see more colour variation.

The **'EQ'** spline remaps the sampled frequency. The default spline does not change the sampled frequency, but suppose we have a spline like this, which remaps low-frequency sounds to zero and high-frequency ones to 2:

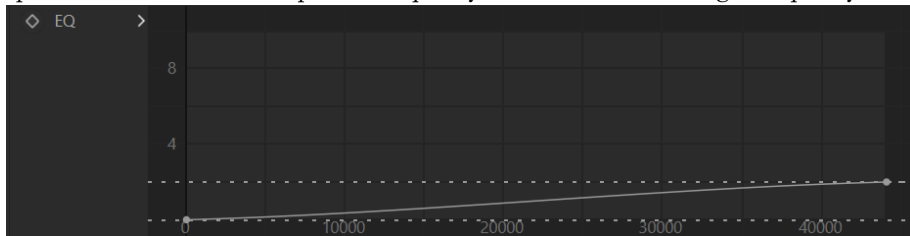


Figure 16.35. Using the EQ spline to remap frequency

This spline will remap low-frequency sounds to zero so that only high-frequency sounds are left. Look at what this does to the graph in the **'General'** quicktab:

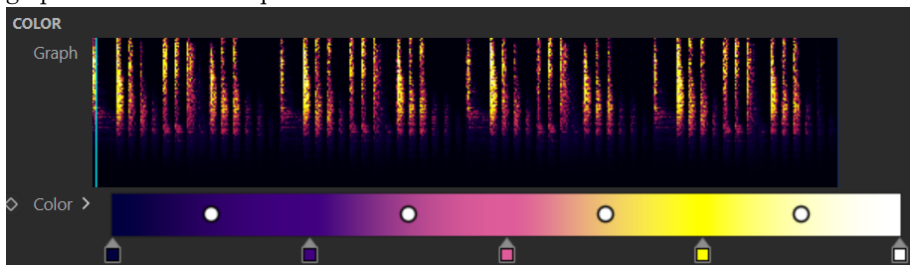


Figure 16.36. Effect of remapping the EQ spline

We've lost all the low-frequency sounds and boosted the high-frequency ones. This gives a result with particle colours shifted to the right of the gradient in the colour map. By increasing the values on the Y-axis of the spline you can use this to boost sounds at a particular frequency rather than reducing them

And that is basically it for the Sound modifier. It seems complex but a little bit of logical thought will show what it is doing and

you should be able to get the results you want. Or, of course, you can just experiment until you get a pleasing result!

Adding a sound track to the timeline

As mentioned earlier the Sound modifier can't actually play the sound. If you want to hear the sound as the scene plays, you will need to add a sound track to the timeline. This is straightforward, just follow these steps:

- Open the Timeline window (Window menu->Timeline (F-Curve)...)
- Add a Null object to the scene and then drag it from the object manager into the Timeline window
- Right-click the Null object in the Timeline window then in the context menu click Add Special Tracks->Sound
- With the Sound track selected, in the attribute manager add the sound file to use to the 'Sound' link field and make sure 'Use Sound' is turned on
- Run the animation and the sound will play; you can turn it off by turning off the 'Use Sound' switch in the track

For clarity, the Timeline window when the above steps are complete would look like this:

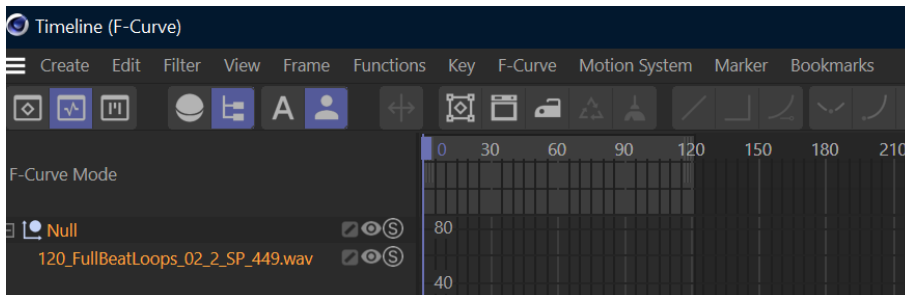


Figure 16.37. Sound track added to timeline

16.16 Trails modifier

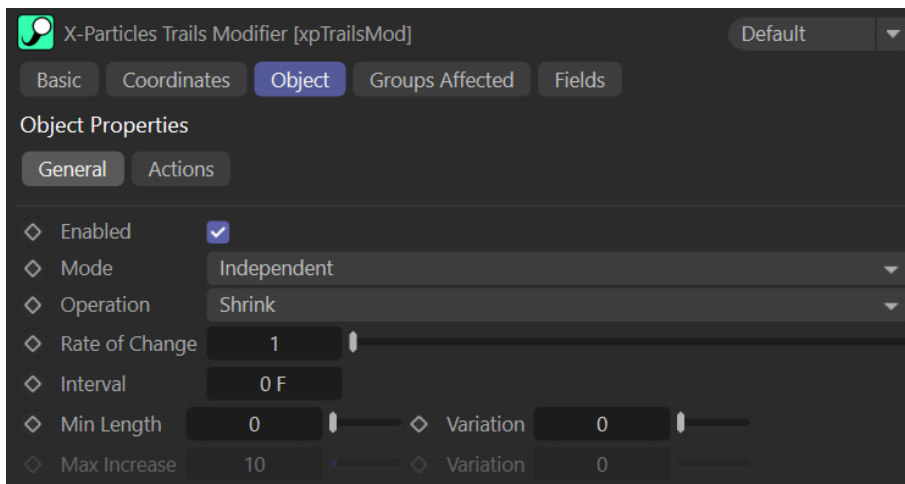


Figure 16.38. Trails modifier

This modifier changes particle trails. A trail is simply a spline and like all splines it is constructed from a series of vertices. In X-Particles, by default a Trail object will add a new vertex each frame, the position of the vertex being that of the particle. The Trail object stores the position of each vertex and then builds a spline from the stored positions.

What it does

Trails are modified in one of four ways. The most common is for trails to shrink, so this is the default mode. When trails are shrunk, what the modifier does is cause the Trail object to build the spline from fewer than the total of stored vertex positions. So for example, if the stored data contains 30 vertices, numbered 0 to 29, it would normally be constructed from vertex 0, which is the first one added to the trail, through to vertex 29, which is the last one added. If the spline is next built from vertex 0 to 28, it will appear to be slightly shorter. When shrinking a trail, the modifier simply reduces the last vertex to use repeatedly, so that over time the spline shrinks. All the position data is still there, however, and after being shortened the trail could be lengthened again.

There are some points arising from this. The first is that shrinking a trail can really only work with the default trail algorithm, **'No Connections'**. The second is what happens if the trail length is increasing each frame and the modifier tries to shrink it. In earlier versions of XP this would cause the trail length to remain the same as it was when the particle entered the modifier field of effect. In the latest version when the trail is affected by the modifier the trail will stop growing and will shrink; this means that you can even shrink trails which otherwise would be actively increasing in length.

That in turn means that you will normally want to have some way to start the modifier working and perhaps stop it again. If you add it to the scene and leave all the defaults as they are, the modifier will start to shrink the trail at the same time the Trail object tries to grow it - and you won't see a trail at all. One way to fix this is to use a field (or falloff pre-R20) so that the modifier only works when the particle is within the field of effect. Another is to use an action to turn the modifier on or off, or you could keyframe the **'Enabled'** switch...and so on. You get the idea.

Why use it?

The main reason is to shrink trails that have already grown. See the example file [ch16_mods_trails_shrink.c4d](#). In this scene, the trails are all set to different lengths. The modifier is inactive until activated by a question which tests whether a particle's trail has reached its maximum length, and if so, activates the modifier for that particle. The question also changes the particle colour, so you can see easily when the trail starts to shrink. If you watch frame by frame, you can see that each trail starts to shrink as soon as it reaches its full length but that other trails keep growing until they also reach their full length.


The modifier has other uses, however. It can expand previously shrunk trails and even grow trails beyond their maximum length. Finally, it can 'pulse' the trails so that they alternately shrink then expand again.

How to use it

This works like any other modifier, but as explained above, you need some kind of control as to when it starts to work on a particle. This could be a Question, an Action triggered elsewhere, or by using a Field object (falloff, pre-R20).

The file [ch16_mods_trails_shrink.c4d](#) shows how to shrink trails. Don't forget the **'Minimum Length'** setting, which lets you specify a minimum length for the shrunk trail. By default this is zero but you can increase it to stop a trail disappearing altogether if required. You can add variation to the minimum length if desired.


Example file [ch16_mods_trails_expand.c4d](#) demonstrates expanding a trail which has previously been shrunk. There are two modifiers in the scene; the first shrinks the trail, then the second expands it again. This is possible because the modifier is non-destructive - it doesn't remove any trail data, it simply moves an internal pointer to the data to be displayed. This also means that the trail won't grown any longer than it did before it was shrunk, as no additional trail data is being added.

 This system will only work if a trail has been shrunk by a Trails modifier. The other way to shrink trails is with an action (see Chapter 13, section 13.7.3). If you do it that way, you won't be able to expand the trail again with this modifier.

For both shrinking and expanding you can control the effect with two other parameters. The **'Interval'** parameter controls the speed of shrinking (or expanding). It is the time interval which must elapse before the next stage of shrinking occurs. The default value is zero, meaning that the trail will shrink each frame. If it is set to 3, for example, three frames must elapse before the trail shrinks again, the result being that the rate of shrinking is slowed.

The **'Rate of Change'** parameter also affects speed but in a different way. The default value is 1, which means that every time the trail shrinks the internal trail data pointer is moved by one vertex in the trail. If set to 3, the pointer is moved by three vertices each time the trail shrinks (or expands). This will make for faster shrinking but it may appear to be jerky because the trail is no longer changing length by one vertex each time. This is, however, the only way to speed up the effect since the **'Interval'** setting cannot be less than zero.

Another operation is to grow the trail. This will increase the trail length even if not previously shrunk, unlike the **'Expand'** operation. See the file [ch16_mods_trails_grow.c4d](#) for an example. Note that once again you can control the effect with the **'Interval'** and **'Rate of Change'** settings, but you also have the ability to set the maximum increase in length (with variation for different particles).

 To use this operation the Trail object must be set to use **'Time'** in its **'Length Mode'** setting. The default is **'Length'** so you must change that before attempting to grow the trails.

Finally, there is the **'Pulse'** operation. This is simply a combination of shrink and expand, which will repeat for as long as the particle is affected by the modifier. File [ch16_mods_trails_pulse.c4d](#) demonstrates this.

Actions quicktab

You can add actions to the **'Actions'** list in this quicktab. These will be triggered in one event only: when the trail reaches a length of zero. If it never does so, the action will never be triggered for that trail's particle.

16.17 Transform modifier

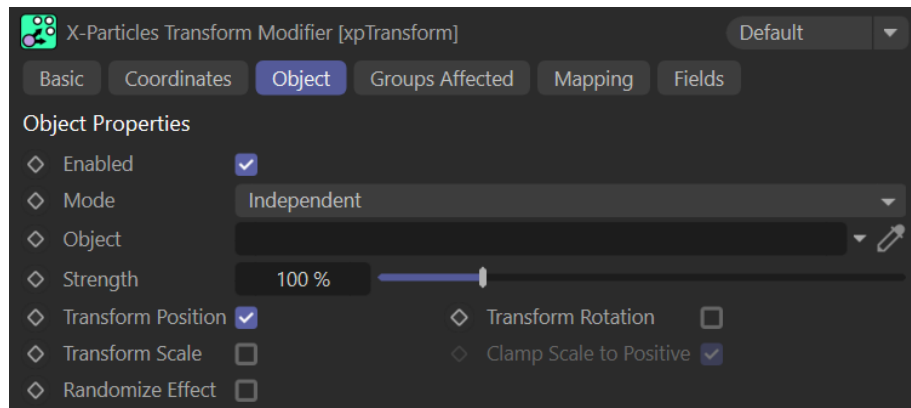


Figure 16.39. Transform modifier

What it does

This modifier transforms the particles (that is to say, changes their position and/or rotation and/or scale) in conformance with another object. For an example, see the file [ch16_mods_transform.c4d](#). This contains a null object with a Cinema 4D Vibrate tag attached. This is set to move the null up and down on the Y axis by 100 units. In the Transform modifier, the null has been dropped into the **'Object'** link field, and the **'Transform Position'** switch is on.

When you play the scene, the null moves up and down and the particles do so as well, showing the same movement as the null object. You can turn on the rotation and scale switches in the modifier as well, and you will see that the particle size and rotation also change.

Why use it?

You might require particles to adopt a certain transformation which you cannot do by using the other tools in X-Particles, but which is very easy to set up by altering the coordinates of a single object. In the example scene a Vibrate tag was used, but you could keyframe the object, bounce it around using C4D's dynamics engine, or whatever method works best, and have the particles follow this movement. This doesn't stop you using other modifiers to transform the particles; you could use a Spin or Scale or Turbulence modifier in the scene and they would still behave as expected.

How to use it

All you need to do is set up the required transforms for the reference object, then drag and drop the object into the **'Object'** link field in the modifier. Set the switches to determine which transforms you want to use and that's all that is required.

There are three other options. The amount to transform is multiplied by the **'Strength'** setting. In the example file, the strength is set to 100%. The Vibrate tag is set to alter the position of the null by 100 units on the Y axis, which means that the null will move up to +100 on Y and down to -100 on Y. The transform modifier, however, transforms the position by twice that amount, due to the way the new position is calculated internally, so the particles will move up by 200 units and down by the same amount.

You can match the movement of the null exactly by reducing the strength setting to 50%. Or you could increase the transform considerably by increasing the strength. This setting gives you additional control over how much the particles are transformed compared to the reference object.

The next option is **'Clamp Scale to Positive'** which is on by default. Although you can set a negative scale for an object this sometimes causes undesirable effects so to prevent that, leave this switch turned on, which will ensure that negative scale values are

treated as positive values. Finally, you can turn on **'Randomize Effect'**. This takes the transform value calculated from the movement of the reference object, then randomly chooses a value between zero and the calculated value. This can result in more interesting effects since the particles will all be transformed slightly differently.

16.18 Trigger Action modifier

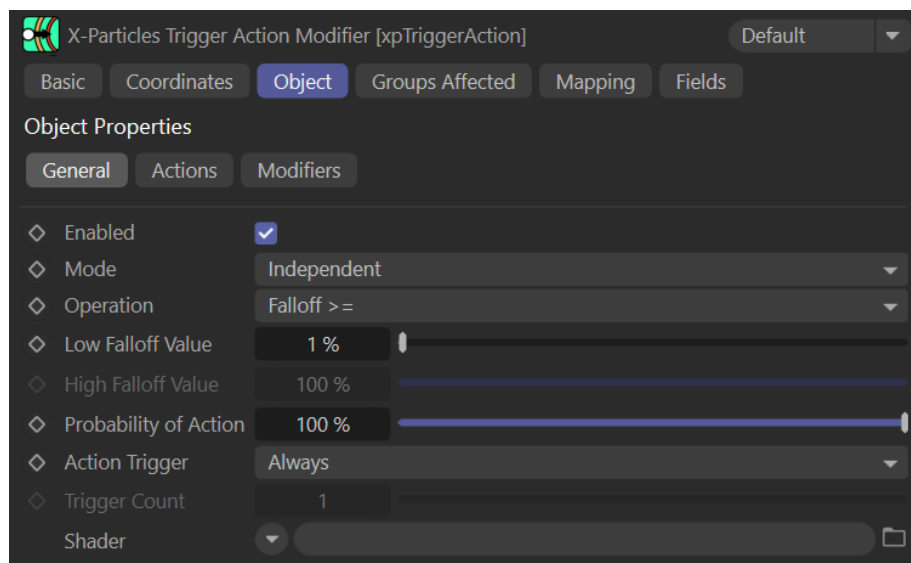


Figure 16.40. Trigger Action modifier

What it does

All this modifier does is trigger one or more Actions, and/or activate or deactivate other modifiers which are in action-controlled mode.

Why use it?

There are many ways to trigger actions in X-Particles, but one way which can be useful is to trigger an action when a particle enters a particular zone in the 3D world. This is the main reason for the existence of this modifier. The action is triggered when the particle enters the modifier's field of effect, so you would almost always use it in conjunction with a Field object (or falloff, pre-R20). Without that the modifier's field of effect is the entire 3D world and the action will be triggered as soon as the particle is created.

How to use it

See the example file [ch16_mods_trigaction.c4d](#). The modifier has a box falloff/field and triggers an action which simply sets the particle colour to white. However, note that it doesn't do so for all particles. At any point within the area of a field or falloff, a value is returned from just greater than zero to one. Outside the area the returned value is zero, in which case the action will never be triggered. For each particle, the modifier takes the falloff value at the particle's position and evaluates whether to trigger the action or not. This depends on the **'Operation'** setting.

If **'Operation'** is set to **'Falloff >='** then the value returned by the field/falloff must be equal to or greater than the value in the **'Low Falloff Value'** setting. If it is not, nothing will happen. In the example file, this setting is set to 85%, so the action will only be triggered if the falloff value is 0.85 (i.e. 85%) or higher. This is why only particles passing through or near to the 'inner zone' of the field/falloff will change colour.

The opposite of this is **'Falloff <='**. Here, the falloff value must be equal to or less than the value in **'High Falloff Value'** for the action to be triggered. Finally, there is the operation **'Falloff Range'**. In this mode, the falloff value must lie between the low and high falloff values.

You will see from the example file that all particles which change colour do so at the same point within the falloff/field. To add some randomness you can set the **'Probability of Action'** value to less than 100%. If the particle meets the falloff values criteria, which would trigger the action, the action will always be triggered if this setting is at 100%. However, reducing this value will reduce the chance that the action is actually triggered. If it is set to zero, it will never be triggered. This probability is retested each frame so if the particle remains in the modifier field of effect for long enough, it is likely that the action will eventually be triggered, but it may not if

the probability value is low and the particle is only in the field of effect for a short time. In the example file, try setting this to a low value - 10% will work fine. You can see that not all particles change colour at the same point, and indeed some never do.

Instead of using the **'Probability of Action'** setting you can add a shader to the **'Shader'** link field. The overall colour brightness from the shader is used as the random probability value. In the example file try adding a Noise shader to this field; you will probably need to reduce the shader's high clip value to around 75% to increase its brightness, or very few particles will change colour. You can even combine using a shader with the probability value if desired.

Another option is to set how many times the action will be triggered. Suppose a particle satisfies the falloff criteria for a number of frames. If **'Action Trigger'** is set to **'Always'**, which is the default setting, the action will be triggered in each of those frames. This doesn't matter if the change is always the same, but it will if the change is cumulative. See the example file [cb16_mods_trigaction_count.c4d](#). This uses a Change Speed action in relative mode, which adds 10 scene units/second to the particle speed each time it is triggered. The action also sets the particle colour to red, so the particles will be red if the action has been triggered at least once for that particle. If you play the scene, you see the red particles move faster than the blue ones and if you use the X-Particles Console to look at the speed, you can see that by the time the red particles leave the field of effect, they have a speed of up to 260 units/second (at a scene frame rate of 30 fps). The initial speed is 150 units/second and the action has increased this each frame.

Now change the **'Action Trigger'** setting to **'For Count'** and leave the **'Trigger Count'** at one. Now the action is triggered just once, so the final speed of the affected (red) particles is 160 units/second.

The rest of the modifier is very simple. In the **'Actions'** quicktab, drag and drop any action you want the modifier to trigger. You can temporarily disable an action by clicking its yellow tick icon to change it to a blue dash. One common use of actions is to enable or disable modifiers in action-controlled mode. You can do this with actions in this modifier, but just as in the Question object, purely for convenience you can drop the modifiers to activate or deactivate into the appropriate list box in the **'Modifiers'** quicktab, without having to use an action. The results are the same either way. See Chapter 12, section 12.5.2 'Activate/Deactivate Modifiers' for more details of how this works in the Question object.

16.19 Unlink TP modifier

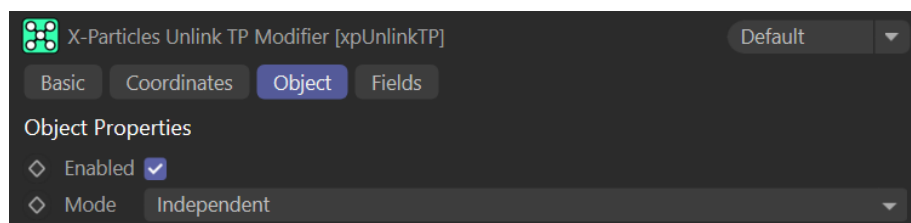


Figure 16.41. Unlink TP modifier

X-Particles can generate Thinking Particles if required. This was added to the earliest versions of XP so that XP could be used with Cinema 4D features such as Pyrocluster (which is still available in C4D). Nowadays there's virtually no call to use TP if you have XP, but the feature still exists.

What it does

When you make the XP emitter generate TP, it generates one thinking particle for each X-Particle in the scene. The TP will follow the XP it is linked to, so as the XP moves, so does the TP. This makes it possible to fake the use of TP with X-Particles modifiers, questions, etc. However, sometimes you may want to release the TP from their associated XP and that is what this modifier does. It uncouples the two particles so that the TPs are now under the control of the TP engine in C4D. Note that this is a one-way function: you can't link the two back together again later.

Why use it?

Thinking Particles are controlled by Xpresso rigs. There are plenty of these available but if you want to use them with TPs generated by an XP emitter, you may have to separate the TP from its controlling XP. You can do that when the TP is emitted (in the emitter's Emission tab, Thinking Particles quicktab, turn on the switch 'Disconnect TP from X-Particles'). The other method is to use this modifier, or there is also an Unlink TP action (see Chapter 13, section 13.5 'Other Actions').

How to use it

Add the modifier to the scene. There are no other settings. Note that you can't select XP particle groups and only unlink those in

specific groups; this modifier affects all particles from an emitter regardless of group.

See the example file [cb16_mods_unlinktp.c4d](#). This generates TP and the linked XPs are shown as blue boxes; you can see the TP at the centre of each box as a small white cross. Normally the XPs are hidden in the scene if TPs are being generated but you can show them again by turning on the switch **'Show Particles'** in the emitter's Display tab, Particles quicktab which is what I have done here.

In the scene the modifier has a Box falloff/field; when the particles enter the box, you can see the TP and XP become separated with the XP still continuing to move under the influence of the Turbulence modifier in the scene, but the TP now move independently and aren't affected by any XP objects. Those which don't enter the field remain linked to the corresponding XP particle.

16.20 Weight modifier

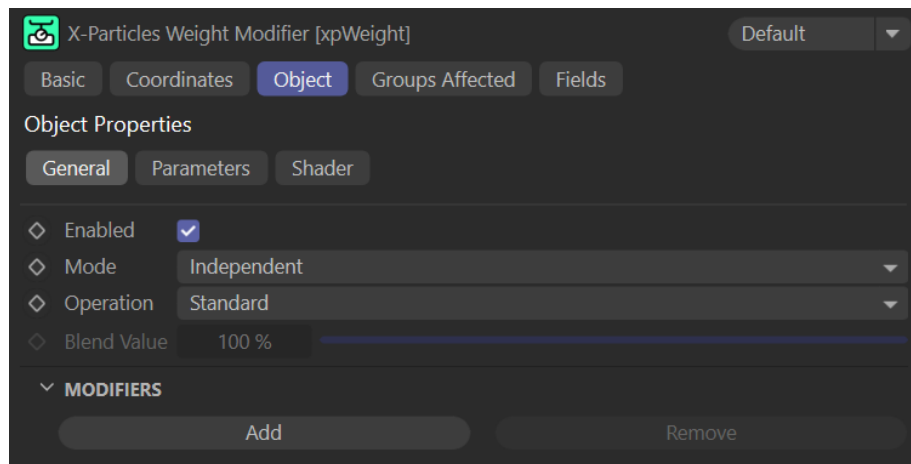


Figure 16.42. Weight modifier

What it does

The Weight modifier 'weights' the effect of other modifiers. It is probably easier to explain this with an example. Take a look at the example file [cb16_mods_weight.c4d](#). This contains an emitter, a Speed modifier, and a (initially disabled) Weight modifier. The Speed modifier simply increases the particle speed by 5 scene units/second each frame. Play the scene and see how the particles rapidly increase speed. Now enable the Weight modifier in the object manager and play the scene again. The particles pick up speed much more slowly.

In the Weight modifier, you see this:

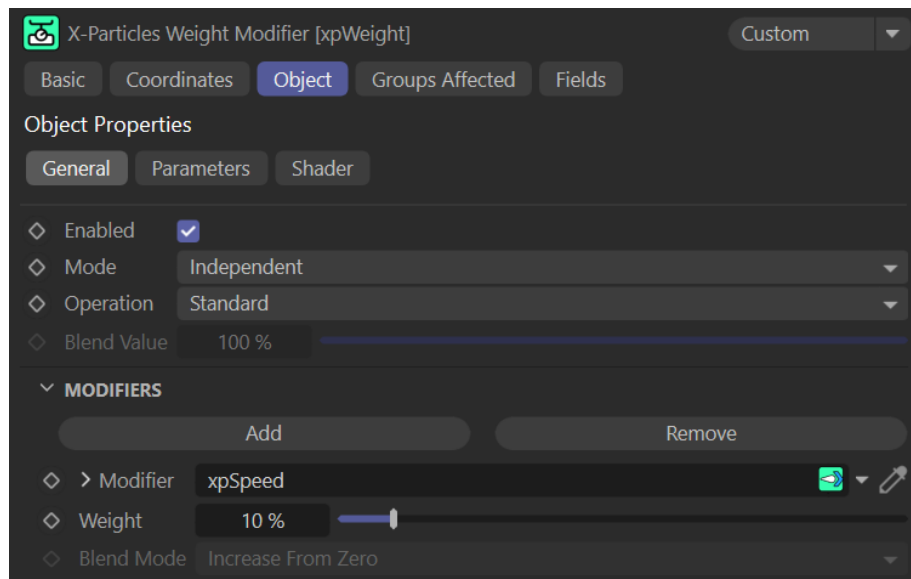


Figure 16.43. Speed modifier added to the Weight modifier

The Speed modifier has been dropped into the **'Modifier'** link field and the **'Weight'** value is set to 10%. What this means is that

the strength of the effect of the Speed modifier - in this case to increase particle speed - is reduced to 10% of what it would otherwise be. Try increasing the **'Weight'** value to 200% and now you see that the particles become very fast much more quickly, because the strength of the Speed modifier effect has been increased rather than reduced.

You can keyframe the **'Weight'** parameter and have the other modifier's strength increase or decrease over time. And you can add multiple modifiers to the same Weight modifier rather than having to create one Weight modifier for each modifier you want to control in this way.

i Not all the effects of all the modifiers will be affected by the weighting. The only way to find out if another modifier can be successfully controlled like this is to try it. The best results are obtained with modifiers which set a continuously variable value such as particle speed or radius, because then the result closely reflects the weight setting. Some modifiers set discrete integer variables and these take some experimentation to get the results you want. See the example file [ch16_mods_weight_gemo.c4d](#). This uses a Geometry modifier to change the index of the object which is produced by a Generator object. The modifier sets the index of the object to generate, which is an integer value. If you play the scene, you can see that the generated objects change from green cubes to red spheres over time. How fast this happens depends on the weight setting in the Weight modifier which in this case affects the probability that the index of the generated object will change.

Why use it?

It's true that in the first example file provided you could get the same effect by changing the **'Speed Value'** setting in the Speed modifier, but not all modifiers are as simple as this, and the effect of the Weight modifier is more to control the overall effect of another modifier than to affect a single parameter. Then if you add in the possibility of controlling the other modifier with another particle parameter, or by a shader, the Weight modifier offers more flexibility and ease of use than adjusting multiple settings in other modifiers to achieve the same result.

How to use it

1. Standard operation

You can weight more than one modifier in a Weight modifier but you have to add each one to the Weight modifier. To do so, click the **'Add'** button which will add this interface:

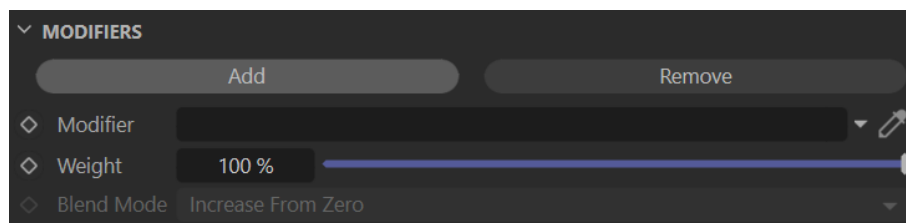


Figure 16.44. Adding a modifier to be weighted to the Weight modifier

Then drag the modifier to be weighted into the **'Modifier'** link field and set the required weight in the **'Weight'** field. Remember that values over 100% can be entered. You can remove modifiers you have added with the **'Remove'** button, but this always removes the last one to have been added; there is no way to remove modifiers in the middle of the list. If you need to remove the effect of the Weight modifier on a modifier in the list, simply set the **'Weight'** value for it to 100%, which will result in no change from the weighting. That's all you need to do to use the Weight modifier in its simplest form but you can do more than this.

2. Blend operation

One of the things you can do is to keyframe the weight setting for each modifier, which could be used to increase or decrease the strength of a modifier over time. However, suppose you are weighting several modifiers, and you want to keyframe the weighting so that some increase in strength while others decrease. This means keyframing several different weight values - and every time you tweak the animation you need to alter all those keyframes. There is an easier way. In the Weight modifier if you change **'Operation'** to **'Blend'** you find that you can no longer adjust the individual weight for each modifier but you can set the **'Blend Value'**. What this does depends on the **'Blend Mode'** setting for each modifier in the list. If that is set to **'Increase From Zero'** then as the blend value increases so does the modifier strength, with a blend value of zero meaning that the weighted modifier does nothing, while a blend value of 100% means the modifier has its full effect. If the mode is **'Decrease From Maximum'** then this is inverted, so a blend value of 100% means that the affected modifier has no effect.

For an example, see the file [ch16_mods_weight_blend.c4d](#). This has a Speed modifier and a Scale modifier weighted using this

method. You can see that the particle speed slows down as you play the scene because the blend mode for the Speed modifier is set to decrease, while the radius increases because the Scale modifier blend mode is set to increase with the blend value. This technique doesn't give you as close control over each modifier but it does make it easier to adjust animations if several modifiers are being weighted, because you only need to keyframe one setting - the **'Blend Value'**.

3. Weighting by parameter

See the example file [ch16_mods_weight_param.c4d](#). In this scene, the particles are spinning thanks to the weighted Spin modifier. The **'Weight'** value for the Spin modifier is not keyframed, but the spin rate decreases as the particle gets older and it will stop spinning altogether when the particle is 150 frames old. This happens because the modifier strength is dependent on another particle parameter, in this case its age. In the **'Parameters'** quicktab you see this:



Figure 16.45. Weighting by parameter (particle age)

What this does is set an age range, 0 to 150 frames (at 30 fps). The spline control governs the Spin modifier strength over this age range. Newly-created particles read the spline from its left hand end, so in this case the spin speed will be maximum for those particles. As the particles age the spline control returns a smaller value until at the age of 150 frames the spline returns zero and the Spin modifier does nothing.

There are several parameters to choose from. As well as age, you can choose lifespan, which acts in the same way but over the full lifespan of the particle (and remember that particles can have different lifespans). You don't need to supply a range for that parameter but for most of the other parameters you must specify a range - and you need to make sure that the range is appropriate to the parameter. For example, a range of zero to 300 scene units is a reasonable start point for particle speed, but that same range for particle radius would probably not work very well; given that radius values are usually small, the control spline would always be sampled at or very near the left hand end. If you choose **'Color'** as the parameter, you don't supply a range. Instead you choose whether to sample the amount of the red, green or blue component of the colour, or the overall colour brightness.

i When we look at data mapping in a later chapter, you will see that this aspect of the Weight modifier is very similar to the data mapping available in many modifiers. The Weight modifier is actually an early version of data mapping used before that feature was added to most modifiers.

4. Weighting by shader

Finally, you can use a shader in the **'Shader'** tab to affect the weighted modifiers. In this case, the overall colour brightness sampled from the shader controls the modifier weight (in conjunction with any other settings used, such as the weight value, parameter settings and so on).

Chapter 17: Modifiers Part 3: Motion Modifiers

The motion modifiers are a group of modifiers whose principle action is to affect particle motion, i.e. speed and direction. You should be aware that X-Particles stores this data as part of the basic particle data, but as a single value - the particle velocity. This is a combination of speed and direction. Speed itself has no direction - the particle could be travelling in any direction with that speed. Conversely, direction doesn't imply that the particle will actually move in that direction; the speed could be zero and the direction simply shows the way the particle would move if it had any speed.

I mention this because particle velocity is stored as a vector value giving the direction of movement, as a combination of movement along the three orthogonal axes, X, Y and Z, with the speed stored as the length of the vector. This is important if you want to use Python scripting or Xpresso nodes to read or change particle velocity. To get the speed, you obtain the length of the velocity vector; to get the direction without speed, you must normalise the vector. If you don't want to use Python or Xpresso you can ignore the above, since the modifiers will do all the necessary mathematics to change particle speed and/or direction.

↓ As always, the example scenes used in this chapter can be downloaded as a single archive, [examples_ch17.zip](#).

17.1 Attractor modifier

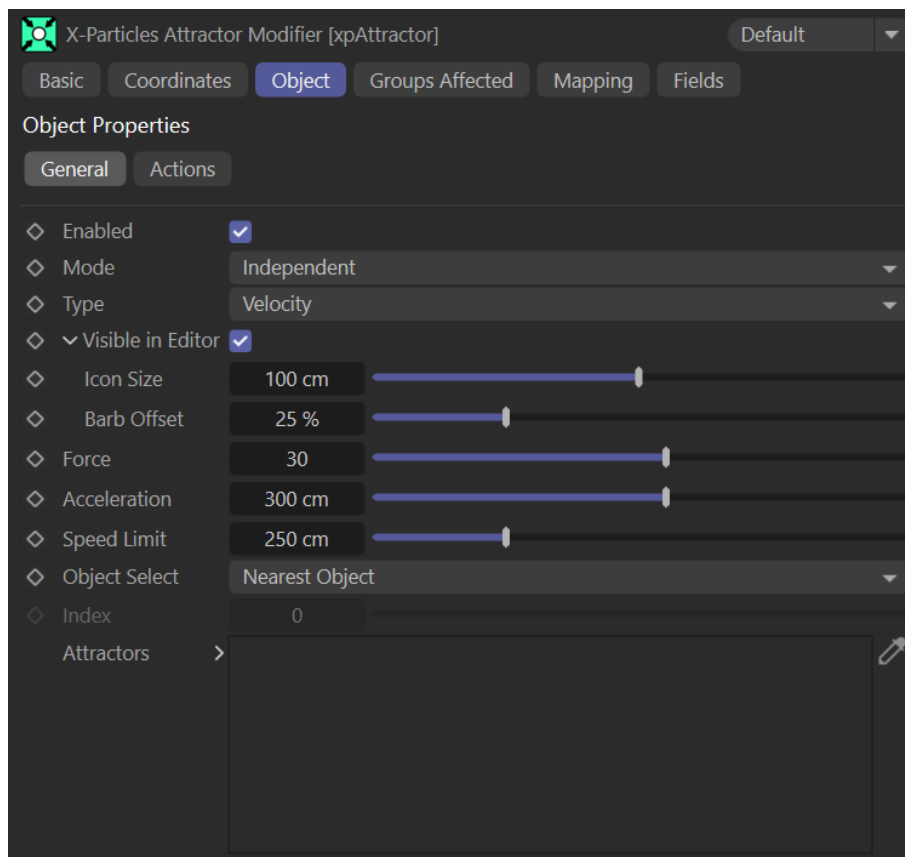


Figure 17.1. Attractor modifier

What it does

As its name implies this modifier attracts particles like a magnet attracts iron filings. It will change both particle speed and direction, the actual result depending on the the settings in the modifier.

This is one of the relatively few modifiers which has an on-screen representation. Figure 17.2 shows how the modifier looks in the viewport.

The four arrows simply point to the centre of the modifier, showing where the particles will head for. You can alter the position of the arrows by changing the '**Barb Offset**' setting, which you can access by clicking the little arrow next to the '**Visible in Editor**' label.

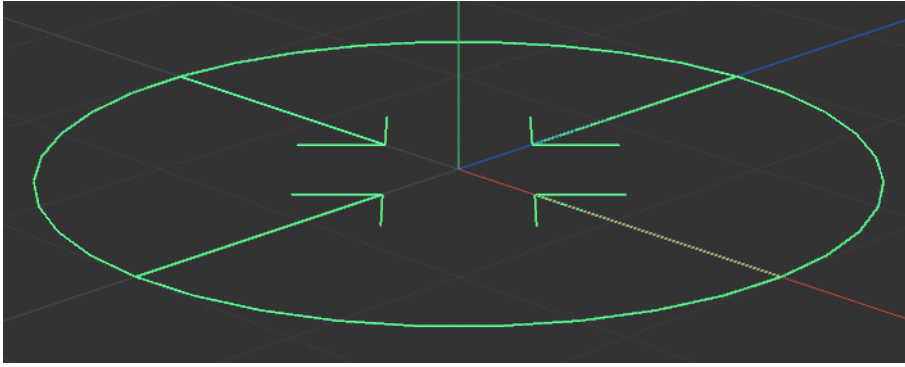


Figure 17.2. Attractor modifier viewport display

You can remove the display if you don't want it by turning off the **'Visible in Editor'** switch, or change the size with the **'Icon Size'** setting.

Why use it?

You can use the modifier to change particle direction, but the result isn't always predictable. If you need particles to move in a specific direction, use a Direction modifier instead. Another use might be to attract particles to an object but not stick to it (for that, use a Cover modifier). Or you can just play around and see what sort of pleasing effects or patterns it will generate.

How to use it

Add the modifier to the scene and by default the particles will be attracted to the modifier's location (where the arrows are pointing). You can see that rotating the modifier will have no effect at all, but the position of the modifier is very important, since that is where the particles will go.

Having decided on location, you should consider which type of attraction to use. This is done with the **'Type'** menu and is really just selecting between two different algorithms, but with one very important difference. If the type is set to **'Velocity'** then the modifier essentially takes over control of particle movement. Other modifiers which change particle direction will have an effect but it will be limited. The other type is **'Force'** and with this, other modifiers have a much more profound effect because the effect they produce will be combined with what the Attractor is doing, rather than trying to carry out their own action independently.

As an example, see the file [ch17_mods_attract_1.c4d](#). This contains two modifiers, an Attractor and a Turbulence modifier (which is initially disabled). Play the scene and note how the particles orbit the modifier in a fairly neat and tidy eclipse. Then turn on the Turbulence modifier. You see that the particles still behave in much the same way, although a little more chaotically due to the turbulence. If you turn off the turbulence, then switch the attractor type to **'Force'**, the particles behave rather differently due to the different algorithm. But if you now enable turbulence again, the added effect of the Turbulence modifier is much greater. Which type you use in the attractor depends entirely on the effect you are trying to achieve, there is no right or wrong way to do this.

Controlling particle attraction

You control the attraction of the particles with three settings, **'Force'**, **'Acceleration'** and **'Speed Limit'**. The **'Force'** is the strength of the attraction; the higher it is, the greater the tendency for the particles to head for the attractor in a straight line. Lower force settings means that they move in a gentler curve. However, it isn't quite as simple as that, because the attraction towards the modifier is also influenced by particle speed, and that in turn is changed by the acceleration. The **'Acceleration'** setting controls how fast the speed increases towards the maximum speed in the **'Speed Limit'** setting.

Getting the effect you want can be a matter of trial and error. See the example file [ch17_mods_attract_spiral.c4d](#). This moves the particles into a nice spiral around the modifier, which is due to a low force and high acceleration. But if you increase the force to 50, the particles move straight to the attractor, overshooting it slightly then moving back to the attractor. You can reduce the overshoot by reducing the speed limit to about 100 scene units. Increasing the speed limit to 300 will result in a greater overshoot, but the high acceleration will still move the particle sharply back to the attractor. In that case, reducing the acceleration to 200 will remove that snapping back and give a nice elliptical pattern as the particles orbit the attractor.

The point of all this is that the three settings interact with each other and you do need to experiment to get the best result.

Attraction to an object

By default particles are attracted to the modifier itself (to its position in the 3D world, to be precise) but you can opt for the particles to be attracted to another object instead. To do this, drag and drop the object into the **'Attractors'** list in the modifier. See the file [ch17_mods_attract_object.c4d](#) for an example; this has a Null object whose position is animated and the particles will always move towards the object's location.

Multiple objects can be added to the list if required. Which one the particles will move to is governed by the **'Object Select'** menu. This will have no effect unless there is more than one object in the list; if there is only one, that object is used, regardless of the menu setting. The first two options - **'Nearest Object'** and **'Furthest Object'** are self-explanatory. Each particle will move towards the object which is closest to (or furthest away from) the particle. Bear in mind that this may change if the position of the objects is animated - another object may become closer, in which case the particles will move towards that object instead. The file [ch17_mods_attract_object_2.c4d](#) is an example where this happens.

The option **'Average'** calculates the mean position of the objects and uses that as the attraction point. In the file [ch17_mods_attract_object_2.c4d](#), change the **'Object Select'** menu to **'Average'** and watch what happens. You can see that the attraction point stabilises between the two Null objects once the animated Null stops moving. **'Object Index'** lets you specify one object from the list, where an index of zero is the first object in the list. If you choose a number which has no meaning (e.g. if there are two objects in the list and you enter an index of 2, which would be the third object if it existed) all the objects are ignored and the modifier itself becomes the attraction point. Finally, you can opt for **'Random Object'** in which each particle moves towards a randomly-selected object from the list.

Actions

In the **'Actions'** quicktab you can add one or more actions to do whatever you like. These actions are triggered when the distance between the particle is either less than the **'Range'** setting (**'Range Mode'** is set to **'Range Less Than'**) or greater than the range setting. Be aware that the actions will be triggered each frame as long as the range criteria are met.

The file [ch17_mods_attract_action.c4d](#) shows this working. The Attractor modifier is in action-controlled mode, but an action is triggered when the particle is close to the modifier's axis, which turns off the modifier and allows the particles to continue moving with their current velocity. You can see when this happens as the action also changes the colour to red.

17.2 Avoid modifier

Figure 17.3 shows the interface of the Avoid modifier.

What it does

The modifier tests if a particle will intersect with a scene object if it carries on its current course, and if so, carries out one or more acts on the particle such as changing direction or triggering an Action.

Why use it

This modifier does not deal with colliding with an object; the X-Particles Collider tag handles that, along with the Particle to Particle Collision object for collisions with other particles. The Avoid modifier doesn't do anything if a particle hits an object, the purpose is to avoid hitting it in the first place. You would use it to prevent such collisions rather than react on collision. Of course, you can detect collisions as well by using the Collider tag, and in fact you might need to do so to handle events if a particle fails to avoid an object.

How to use it

Take a look at the example file [ch17_mods_avoid_objects.c4d](#), which demonstrates several aspects of using this modifier. Any objects the particles should avoid must be dropped into the **'Objects'** list and you can temporarily ignore objects in the list by clicking the yellow tick so it becomes a blue dash.

In the scene there are two objects to be avoided, a Cube and a Star spline, but the spline is initially disabled and ignored in the list (blue dash icon). Play the scene and watch what happens. You can see that the particles turn green when they detect the Cube due to the action in the **'Actions'** list, but that some cannot avoid it completely and turn red because they collide with it. The particles detect the object when they are 50 scene units away from it (the **'Detection Distance'** setting) - but away from what? Not simply the

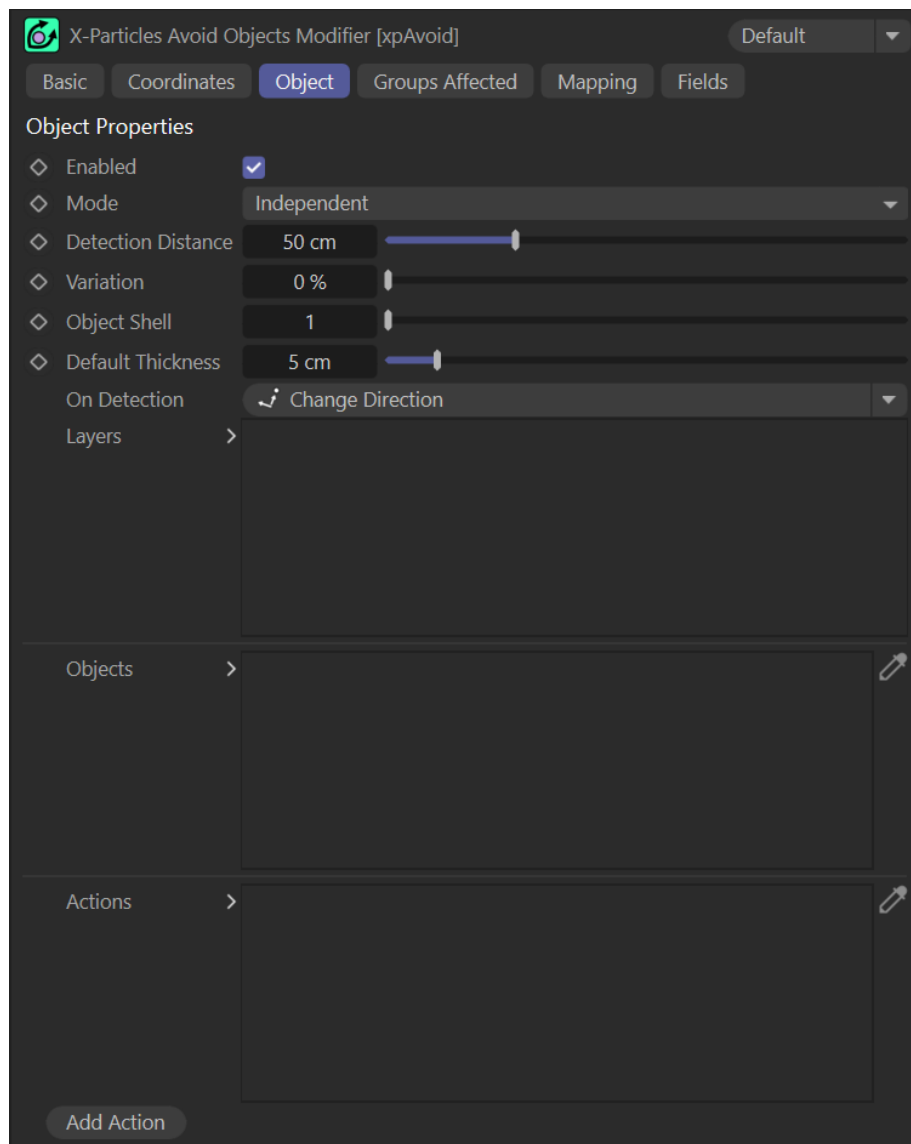


Figure 17.3. Avoid modifier

position of the object in the 3D world, which would be pretty useless, but the actual surface geometry. If you increase the **'Detection Distance'** to 100 scene units, you can see that they start to move when they are further away, but even so, a few particles will still collide with the object and turn red.

This happens because the modifier tries to keep the deflection of the particle to a minimum and it doesn't always avoid the object completely. Increasing the detection distance will (mostly) fix that problem, but you might not want to do that. Instead, you can increase the **'Object Shell'** value. This simulates a shell around the object and the modifier uses that instead of the actual object geometry. You often don't need to increase this much at all. In this scene, with a detection distance of 50 scene units, a shell setting of 1.1 is all you need to prevent any collisions.

Don't make the **'Detection Distance'** setting too small. If you do, some particles may not be able to react in time to avoid collisions. Even if there are no collisions, the change of direction to avoid the object may be so severe that it looks as if collisions are occurring. You can add some per-particle variation to the detection distance with the **'Variation'** setting.

The next thing to do is decide what you want to happen when the modifier detects a possible collision with an object. You do this with the **'On Detection'** menu. Selecting an option from this menu will add one of four possible effects into the **'Layers'** list. Unless this list contains at least one enabled entry, the modifier will do nothing.

Figure 17.4 is a screenshot from the example file showing the list with all four entries in it. You can see that two (**'Change Direction'** and **'Trigger Actions'**) are enabled because their switch is turned on, while for the other two (**'Freeze'** and **'Die'**) it is off and they are currently disabled. **'Freeze'** and **'Die'** are self-explanatory. If these are active the particle will either be frozen in place or

killed and removed from the scene. **‘Die’** takes precedence over all the other effects; if this is present and enabled, the particles will die no matter what other effects are in the list. You can see these both working by turning on the switch to enable them.



Figure 17.4. Possible effects when an object to be avoided is detected

The **‘Trigger Actions’** effect will trigger any actions in the **‘Actions’** list. There can be as many actions as you like and as with the objects, you can temporarily disable them with the yellow tick/blue dash icon.

There is one other setting to discuss and this is the **‘Default Thickness’** setting which is used when avoiding splines. In the example, scene, disable the Cube in the object manager and enable the Star spline. In the modifier ignore the Cube and avoid the Star instead like so:

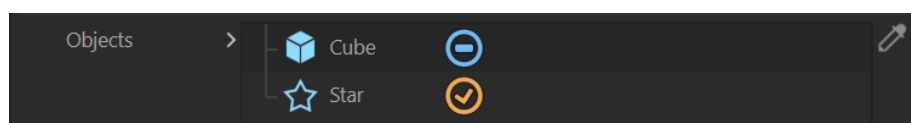


Figure 17.5. Avoiding a Star spline instead of a Cube object

If you play the scene, you can see that nothing happens. The problem is this: since a spline has no polygons, how can any particle collide with it? The Avoid modifier gets round this by assuming that the spline has a virtual cylindrical ‘shell’ around it, and the diameter of this virtual shell is found in the **‘Default Thickness’** setting. If you increase this to 10 scene units, you can see that the modifier detects the object and some of the particles change direction. As you increase the thickness, more particles will be affected until at about 15 units, they all are.

Changing direction

The primary aim of this modifier is to prevent collision with a scene object, which it does by changing the particle’s direction so it doesn’t hit the object. But that change in direction is permanent; see the file [ch17_mods_avoid_chgdir.c4d](#). When the particles avoid the sphere they change direction and that new direction is very different from the initial one. Suppose you want the particles to move in their initial direction after avoiding the sphere. There is no simple way to do this but the following trick may sometimes be useful. In the scene file, turn on the Attractor modifier and see the result. This happens because the particles are initially drawn to a Null object by the Attractor, then they avoid the sphere, and then they continue to head for the Null object. Since this is a long way away (10,000 scene units) from the emitter, the particle paths appear to be parallel to one another. This is only of value if your particles are all heading in the same general direction, but you can use other methods to mitigate this. In the scene file, turn off the Attractor and turn on the Cover modifier and the Plane object. Now the particles are moving in different directions (due to different target points on the Plane object) but after avoiding the sphere, they continue to move to the same target point as before.

Avoiding particle trails

Why is the thickness setting called **‘Default Thickness’**? This is because this feature was added initially so that particles could avoid particle trails, either their own or those of other particles. Look at example file [ch17_mods_avoid_trails.c4d](#). This has two emitters with the green particles also having trails, and we want the blue particles to avoid the trails. If you play the scene, note first of all that the ‘Trail Emitter’ must not be affected by the modifier, and to ensure this the modifier is dropped into in the **‘Objects’** list of the ‘Trail Emitter’ modifiers tab. When you play the scene, you see that very few blue particles avoid the trail splines (the ones which turn red), and increasing the **‘Default Thickness’** makes no difference at all. The reason is that in the case of XP trails, the trail thickness is stored in the trail data and in turn is taken from the ‘Thickness & Color’ settings in the Trail object (see Chapter 11, section 11.2 ‘Thickness and Colour’). The default option for the thickness is **‘Do Not Set Thickness’** but the thickness data is still present and with this option it is assumed to be 0.5 scene units. It is for this reason that only the few particles which come very close to the trail splines are affected and why altering the default thickness in the modifier does nothing.

There are two possible solutions to this. In the Trail object, you can set the **‘Thickness Mode’** to another option, such as **‘Set From Value’**. If you do that in the example file, then you see a considerable difference and the thickness of the spline is now controlled by the Trail object **‘Thickness Value’** setting. The other option is to turn on the switch **‘No Thickness or Color Data’** in the Trail object.

Doing this will mean that there is no data in the trails to use for thickness, in which case the modifier uses its **‘Default Thickness’** setting instead.

There is another problem with trails, however. This is seen when a particle is trying to avoid its own trail. See the file [cb17_mods_avoid_own_trail.c4d](#). This contains a Network modifier, but you can ignore that - it’s only there to make the particle change direction and (eventually) have to avoid its own trail. Play the scene, and all works as expected, the particle avoids the trail, usually by moving up or down the Y axis to do so. Now in the Trail object, switch the thickness to **‘Set From Value’**, which is set at 6 scene units. Play it again and watch what happens!

The reason for this is that the spline thickness is conceived as a virtual ‘shell’ around the particle. Each frame, the modifier calculates where the particle will move to in that frame and if it would then hit the shell around the spline. With the thickness mode set to **‘Do Not Set Thickness’** then as mentioned above the shell is assumed to have a diameter of 0.5 scene units. In this scene the particle will move 5 scene units in whatever direction it has, so the shell will never be hit. But if the thickness mode is **‘Set From Value’** and the value is 6 units, the calculated position of the particle will still be inside its shell and the modifier will assume the particle will hit the trail. This happens every frame, hence the constant change of direction that you see. If you reduce the value to 4 scene units the problem will disappear because now the shell is smaller than the distance that the particle will travel. You can also try turning on **‘No Thickness or Color Data’** in the Trail object and altering the **‘Default Thickness’** in the modifier. However, avoiding a particle’s own trail is not guaranteed and you may still see anomalies especially if setting the trail thickness with a lot of variation.

17.3 Cover/Target modifier

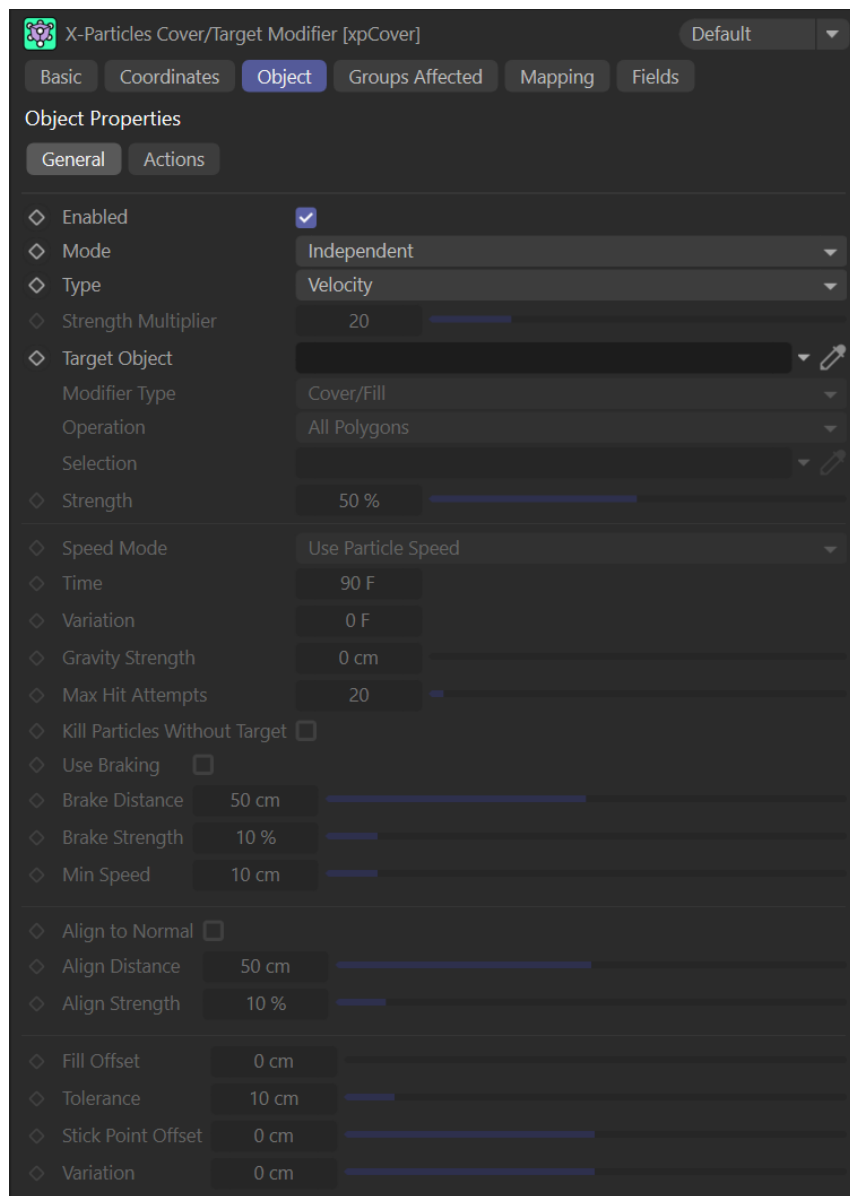


Figure 17.6. Cover/Target modifier

What it does

This is one of the most widely-used modifiers in the X-Particles suite. Its main purpose is to cover an object, or part of an object, with particles, which stick to the object if it is animated. There are a large range of options when using this modifier, but the basic principles of use are very simple.

It also has a secondary function whereby a point on an object can be used as a target for particle movement without the particles sticking to it. In this mode it acts in a similar way to the **Attractor** modifier, but whereas that modifier only attracts particles to an object's position in the 3D world, the **Cover** modifier attracts them a specific point on (or in) the object.

Why use it?

You would use this modifier to attract particles to an object - either a point on the surface or in an object's volume - and optionally stick the particles to that point. You can use it to coat an object, perhaps making it visible when it was initially invisible, or fill an object with particles.


How to use it


There are a lot of options in this modifier, but you won't need to set them all - some are only for specific effects.

The first thing to decide is if you want the particles to stick to the object or not. If you do, the **'Modifier Type'** menu must be set to **'Cover/Fill'**; if not, then set this to **'Target'**. Most other options are available for either type, with a few differences. Most of the time you will probably want the cover/fill option.

Then you can set the **'Type'** option, which can either be **'Velocity'** or **'Force'**. If the type is set to **'Velocity'** then the modifier essentially takes over control of particle movement. Other modifiers which change particle direction will have an effect but it will be limited. The other type is **'Force'** and with this, other modifiers have a much more profound effect because the effect they produce will be combined with what the Cover modifier is doing, rather than trying to carry out their own action independently. You can leave this at the default **'Velocity'** type in most cases. If you change to **'Force'** you also have a **'Strength Multiplier'** setting which increases the strength of the modifier's effect. The more you increase this, the more the modifier will behave as if **'Velocity'** was selected as the type, so you can use this to adjust the combination of this and other modifiers' effects.


There is another setting to control the strength of the modifier - the **'Strength'** setting. This is a measure of how strongly the particles will be pulled to their target position. If this value is low, the particles move in a gentle curve to their target, but with high values they move directly to the target location.

 If this value is set too low, it may cause particles to miss their target altogether.

 In previous versions of XP, this setting was known as 'Acuteness of Turn'.

The above options can all be left at their defaults, but there is one thing you must set: you need to supply an object to act as the target by dragging and dropping an object into the **'Target Object'** link field. Whatever object you drop in here must make sense for the operation to be carried out - for example, if you want to use the object's polygons as the target, it must actually have some polygons!

This leads to the next setting, which you could leave at the default but in many cases you will want to change it. This is the **'Operation'** setting, which is where you specify where the particles should go.

 For what happens if you select an **'Operation'** mode which isn't appropriate for the object - for example, a polygon mode when the target is a spline - see the 'Selections' section below.

Polygon operation modes

The first five entries in this menu all relate to an object's polygons. The default is 'All Polygons' which will send particles to randomly-selected points anywhere on the polygon surface. This can be refined slightly by choosing **'Polygon Center'** which will still use all the polygons but only their centre point.

'Nearest Surface Point' is a useful mode, because it does exactly that - directs particles to the nearest point on the object's surface.

If you select this option, three more settings become available. The first is **‘Threshold’**. See the example file [ch17_mods_cover_threshold.c4d](#). The object for the modifier is a Plane with 19 x 19 polygons - this was chosen deliberately to demonstrate this option. Because the emitter is very small, and both it and the Plane have X and Y coordinates of zero, we would expect the nearest surface point to be somewhere on the centre polygon. However, when you play the scene you can see that points on neighbouring polygons are also covered, which isn't what we want.

The way this feature works is that it first selects the closest polygon and then finds the closest point on that polygon. The problem is that on the Plane there are 8 polygons which are almost as close as the centre one. This can lead the modifier to choose one of those, and then finding the closest point on that polygon - which is why you see the slightly odd pattern of particles. To prevent this, the **‘Threshold’** setting can be lowered, forcing the modifier to be more precise in selecting the nearest polygon. In this scene you will need to reduce the threshold to 0.25 or lower to remove this issue.

The second additional option is **‘Max Hit Attempts’**. We've already seen that the Cover modifier in a polygon mode first chooses the nearest polygon to the particle, then chooses the closest point it can find on that polygon. The problem is that any polygon has an infinite number of possible surface points to choose from. Clearly, it isn't possible to test an infinite number of points to find the closest one, so the modifier tests a number of randomly-selected points and chooses the closest one from that set of points. The number tested is defined in the **‘Max Hit Attempts’** setting. The more random points it tests, the greater the accuracy will be - that is, the nearest the selected point will be to the actual closest point. You can see this in the same example file [ch17_mods_cover_threshold.c4d](#). Reduce the threshold to 0.25 and all the particles stick to a point in one polygon, but you can see that they cover almost all of that polygon. The emitter itself is very small so you'd expect the area covered to be no bigger than that. If you increase the **‘Max Hit Attempts’** value to 25, the area covered is noticeably smaller; increase it to 20 and the area is little larger than the size of the emitter. This would be a good setting for a cutting torch or laser beam effect, while a lower value would be better suited to a paint spray can.

Finally, this mode also has a switch named **‘Repeat Sample’**. If this is off, the nearest point will be calculated just once, when the particle enters the modifier's field of effect. If it is turned on, it will be recalculated each frame. You would use this if the object was being animated, which would probably cause the nearest point to a particle to change as the animation proceeds.

The remaining two polygon modes are intended to stick particles only to polygons which are visible from the particle. There are two ways to do this. The first, **‘Visible Polygons (Using Normals)’** uses the surface normal of a polygon to decide if it is visible from the particle. If the normal is at an angle of less than 90 degrees from the particle position, it is assumed to be visible, otherwise it is invisible. This method is fast but not wholly reliable. See the example file [ch17_mods_cover_vispolys.c4d](#). Let this play right to the end and you can see that there are two problems. Some polygons on the back of the torus, which clearly aren't visible to any particle, attract some particles, while there is a patch of visible polygons on the front of the torus which receive only a few particles:

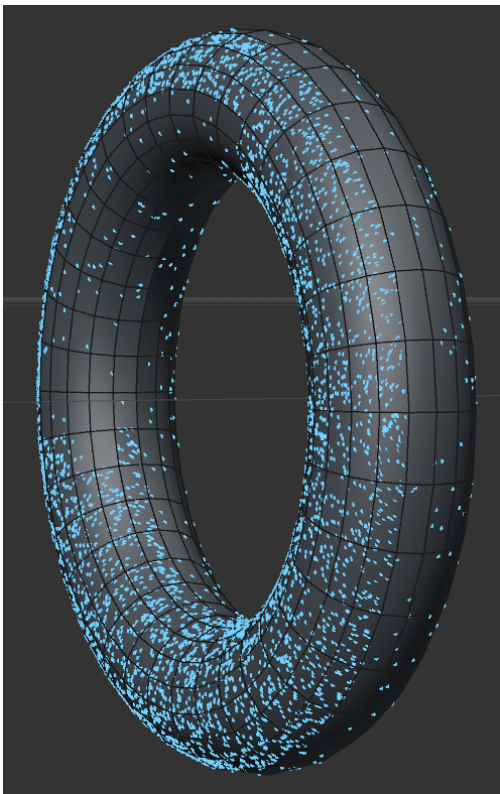


Figure 17.7. Visible polygons mode showing errors

The first problem is due to inbuilt arithmetical error and you can often get round this by increasing the **'Max Hit Attempts'** value. In the file this is set to 5, and if you increase it to 50, the problem disappears. The second problem occurs because the bare patch has reversed normals and so by this algorithm are invisible. Increasing **'Max Hit Attempts'** will not fix this. If you encounter these problems then the solution is to use the other method, **'Visible Polygons (Using Rays)'**. What this does is fire a series of invisible rays from each particle. If a ray hits a polygon, even one with reversed normals, it is visible to the particle. If you reload the scene and change to the rays method, there is no longer a bare patch and the polygons on the back of the torus no longer attract any particles.

However, this demonstrates something else to be aware of. What happens if - whatever **'Operation'** mode is chosen - the particle cannot find a suitable target? In that case the modifier will try again in the next frame and will keep on doing so in successive frames until it succeeds. This could lead to some particles never finding a target and moving freely in the scene. The other possibility is that the animation will slow down because the modifier is constantly trying to find a target for particles without one. If you play the example file, still using the rays method, and with **'Max Hit Attempts'** set to 5, you see that during the animation 6000 particles are emitted and in this case all eventually find a target. Now turn on the switch **'Kill Particles Without Target'**. The same 6000 particles are emitted but those which do not find a target at the first attempt are removed from the scene. This switch can be very useful if you find you have a lot of particles which never go to a target point. Of course, you can also reduce the chance of this happening by increasing **'Max Hit Attempts'** which gives the modifier a better chance of finding a target at the first try. In this file you would need to increase this to around 15 to prevent any particles from being killed.

⚠ Note that this switch can be used in any of the **'Operation'** modes, not just the polygon modes.

Vertex operation modes

The next four entries in the **'Operation'** menu all stick particles to an object's vertices rather than polygons. You can try out these modes using the example file *ch17_mods_cover_vispolys.c4d* - just change the mode in the **'Operation'** menu.

The first option is **'Object Vertices (Random)'** which, as it says, selects a random vertex for each particle. The opposite to this is **'Object Vertices (Ordered)'** in which the first particle sticks to the first vertex, the second particle to the second vertex, and so on. If there are more particles than vertices this method simply wraps round to the first vertex again. **'Object Vertices (Ordered, Random)'** is a combination of the two first two modes. In each frame, all the vertices will receive at least one particle (assuming there is a sufficient number of particles) but the order in which the vertices receive a particle is randomly selected. The final option is **'Nearest Vertex'** which simply means that the particle sticks to the nearest vertex to it. One point to note though is that for some objects there may be multiple vertices at the same distance from the particle, in which case one of those vertices will be selected randomly. You can see what happens if you select this mode in the example file.

⚠ The **'Max Hit Attempts'** setting is not applicable to any of the vertex modes and is not available if one of them is selected.

Object volume mode

This mode will choose a random location within the enclosed volume of the object. If the object isn't completely closed, this may or may not work correctly or at all. See the file *ch17_mods_cover_objvol.c4d* as an example of an incompletely closed volume.

There is an additional setting for this mode - **'Fill Offset'**. This will offset the target position towards the axis of the object by the distance given in this parameter.

Texture mode

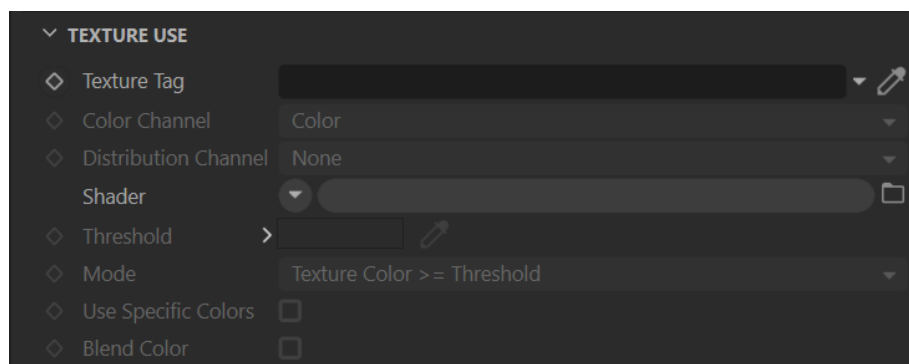


Figure 17.8. Texture mode interface

This mode enables control of particle distribution on the object by use of a texture. There are two ways to do this. The first is to use a shader, which is added to the **'Shader'** link field. What happens then is that for each particle the shader is sampled at a randomly-chosen point in the shader space and the brightness of the colour at that point is tested against the **'Threshold'** colour. Depending on the **'Mode'** setting, that point either receives a particle or it does not, in which case the modifier tries to find another point for that particle. See the example file *ch17_mods_cover_shader.c4d*. This uses a Noise shader with **'Mode'** set to **'Texture Color >= Threshold'** and the **'Threshold'** colour set to mid-grey. This means that the white areas in the shader receive particles because the brightness is greater than that of the threshold, whereas the black areas do not. If you change the mode to **'Texture Color <= Threshold'** the situation is reversed and it's the black areas which receive particles.

i The shader is sampled in shader space, meaning that the position of the object in the 3D world is irrelevant - you will get the same result regardless of object position and rotation. Also, be aware that the shader is sampled using UVW coordinates, so in the example file, if you look carefully you can see that each face of the Cube has exactly the same particle distribution, which is due to the UV mapping of the Cube. A different UV map would give different results.

! If the threshold colour is set to pure white, and the mode is set to **'Texture Color >= Threshold'** only white points in the sampled shader colour will receive a particle. However, if the threshold colour is pure black, all points will receive a particle.

There are two other controls relevant here. By default the overall brightness of the shader sample is tested against the overall brightness of the threshold colour. Try this: in the same example file, go into the Noise shader and change the white colour to pure red, then change the threshold colour to pure blue. Play the scene and it works as expected, because although the colours are different the overall brightness of pure red is the same as that of pure blue, so the test **'Texture Color >= Threshold'** is passed and that point will receive a particle. Now turn on the switch **'Use Specific Colors'** and replay the scene. No particles will now stick to the Cube. This happens because turning on the switch forces the modifier to test the red, green and blue components of the sampled colour individually against those of the threshold colour. Each component must pass the test for the sampled point to receive a particle, and that won't happen when comparing pure red (RGB 1, 0, 0) to the pure blue threshold (RGB 0, 0, 1).

The other useful control is **'Blend Color'**. If this is turned on, as the particles head to their target point, the particle colour will be blended with the sampled colour until the particle sticks to its target point, when the particle colour be the same as the sampled colour. See the example file *ch17_mods_cover_colourblend.c4d*. This uses an abstract bitmap from Pixabay (<https://pixabay.com>) in the **'Shader'** link field, rather than a procedural shader. The **'Threshold'** colour is set to black so that all sampled points receive a particle. If you run the scene, the results are unremarkable. But now turn on **'Blend Color'** and watch what happens. This is the final frame in the animation:

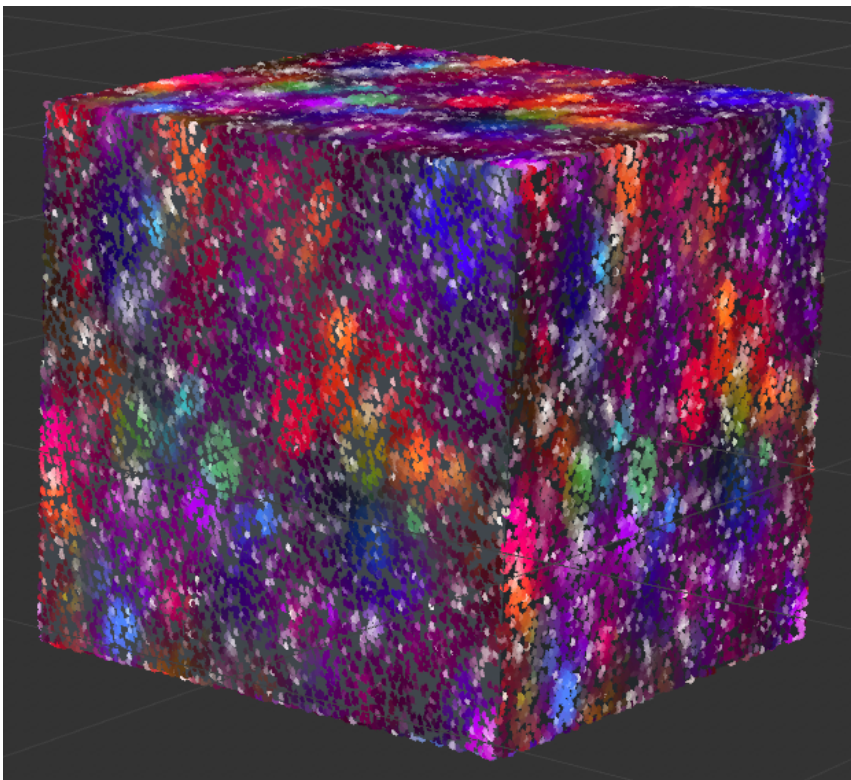


Figure 17.9. Final frame in **'Blend Color'** animation

You can see that the effect of **'Blend Color'** is to reproduce in particles the pattern and colours from a shader or bitmap onto the target object.

Instead of using a shader, you can leave the shader link empty and instead drag and drop a texture tag into the **'Texture Tag'** link field.

⚠ You can't use a shader and a texture tag at the same time. Adding a texture tag will make the shader link field unavailable; if there is already a shader in the link, it will be ignored, as the texture tag will take precedence.

Adding a texture tag gives access to two more settings - **'Color Channel'** and **'Distribution Channel'**. The colour channel refers to the channel of the material which, by default, will control particle distribution on the surface and the final particle colour if **'Blend Color'** is turned on. The default setting is to use the Color channel of the material, but it doesn't have to be - you can use any of the channels in the drop-down list. The channel you select does not have to be active in the material, you can set up any of these channels and use it even if it doesn't actually contribute to the overall texture.

However, it is possible to control the distribution with a second channel specified in the **'Distribution Channel'** setting. Again, you can choose any channel from the menu, active or not in the material. If this menu is set to anything other than **'None'** the particle distribution (but not the colour) will be taken from the specified channel. To see how this works, try the example file [ch17_mods_cover_ft_distrib.c4d](#). Play the scene and you see it works exactly as expected. You can turn on **'Blend Color'** and the particle colours are set accordingly. Now change the distribution channel to 'Diffusion' and play the scene. The diffusion channel in the material contains a Noise shader to control particle distribution, but not the particle colour, as you can see if **'Blend Colour'** is turned on; the colours are still taken from the material Color channel. Note also that the Diffusion channel of the material is not active and does not contribute to the final texture.

Finally, in this example the texture tag comes from the target object for the particles, but again, that isn't mandatory. You can use a texture tag from a different object. In the same example file, drag the texture tag from the invisible Cube object into the link field. The material in that tag contains a Fire shader in its Color channel, and you see the particle distribution (and colour, if **'Blend Color'** is turned on) is controlled by that texture, not the one on the target object.

Particle speed control

You can set the speed with which the particle travels to its target with the **'Speed Mode'** setting. The default setting is **'Use Particle Speed'** and with this the particles will keep the speed they already have. This will cause the particles to arrive at their targets at slightly different times, depending on how far away they were initially from their target point. If you want them all to arrive at the same time, choose **'Set Time to Reach Target'**. Then each particle's speed will be adjusted to arrive in the same length of time as given in the **'Time'** setting. You can add variation to this if desired with the **'Variation'** setting. Finally there is **'Gravity'** mode, which accelerates the particle as it is pulled to the target point. This is adjusted with the **'Gravity Strength'** setting. Note that only small values (5-10 scene units) are usually required, and if the strength is too high, the particles may overshoot their target point.


Braking

Without braking the particles will move to their target point and abruptly halt when they reach it. You can force them to slow down when approaching their target by turning on the switch **'Use Braking'**. With that enabled, the particles will start to slow down when they are the distance given by **'Brake Distance'** from their target, and each frame their speed will be reduced by the percentage in **'Brake Strength'**. To stop the particles becoming unacceptably slow, you can set the **'Min Speed'** value, and the speed will not fall below this.

Align to Normal

If you have a particle with a definite axis of direction to it, rather than a simple dot or sphere, you might want the axis to be aligned with the surface normal when the particle sticks to the object. In the example file [ch17_mods_cover_align.c4d](#), there is a Generator object producing cones whose axis is along the Z axis. If you play the scene, the results aren't very pleasing because when stuck to the sphere the cones still point along the Z axis. Now turn on **'Align to Normal'** and replay the scene. The cones start to rotate as they near the target point and end up pointing along the polygon normals. As with braking, you can set the distance at which they start to align (**'Align Distance'**) and the percentage by which they rotate into alignment each frame (**'Align Strength'**).

Note that this feature will work in the vertex modes as well as the polygon modes, but only if the target object is a polygon object. Aligning the particles will not work with splines.

 This works by rotating the particles and therefore you must turn on **'Use Rotation'** in the emitter's **'Extended Data'** tab. You don't need to set anything else there but rotations must be enabled.

Other settings

There are some other parameters which haven't yet been covered. **'Tolerance'** is the distance at or below which the particle will snap to its target point. This is there to avoid the problem that a particle may never actually arrive at its target position - each frame it may overshoot the target and oscillate around it for ever. If you set this too low, this is exactly what will happen, but if you set it too high, the particles may make a sudden and very obvious jump to the target location. How high this needs to be depends on the particle speed (higher speeds may need a higher tolerance) and the effect you are trying to achieve.


'Stick Point Offset' adds an offset distance to the target location. Positive values will make the target point 'outside' the object while negative values draw the particles inwards. You can add variation to this with the **'Variation'** setting.

Actions

You can add Actions which will be triggered when the particle sticks to its target (in **'Cover/Fill'** mode) or when it comes within a specified range of the target (**'Target'** mode). Drag and drop the required Actions into the **'Actions'** list. As usual, they can be temporarily disabled by clicking the yellow tick icon to change it to a blue dash. From the **'Repeat Actions'** menu you can choose whether you want the actions to be triggered only once, when the particles stick (**'No Repeat'**) or if they are to be triggered in subsequent frames as well (**'Repeat'**). Finally, in **'Target'** mode only, you can set the range when the actions are to be triggered in the **'Range'** setting.

Selections


You don't have to use all the polygons or vertices in an object as targets for the particles. You can restrict the targets to those areas you want by using a selection.

 You can only use polygon or point selections. Edge selections are not accepted by the modifier.

Drag a polygon or point selection tag into the **'Selection'** link field. The modifier will do its best to convert between points and polygons if required. In other words, if you select one of the vertex modes and supply a polygon selection, that will be converted into a point selection. Likewise, in one of the polygon modes a point selection will be converted to a polygon selection.

If the target object is a spline, there cannot be any polygon selections, only point selections. If you use a polygon mode with a spline object, the modifier will recognise this and automatically switch to the random vertices mode; point selections will then be honoured as they would if you were using a vertex mode.

The other possibility is that the target object is a polygon object with no polygons, just vertices. The vertex modes will all work as expected, including point selections, but none of the polygon modes will work at all - that is, the modifier expects to find polygons but can't find any, so has no effect on the particles.

 Selections cannot be used in either of the visible polygon modes (using rays or normals) and will not work in texture mode either. For obvious reasons **'Object Volume'** mode also cannot work with selections.

Preventing particles from moving through an object

One thing which may occasionally be a problem is that in order to reach the target point particles often pass right through an object. This isn't always desirable, so how can it be avoided? See the example file [ch17_mods_cover_followsurface.c4d](#). When you play it, you can see that the particles heading for the far side of the sphere pass through the sphere's volume. Now enable the Follow Surface modifier in the object manager. Now those particles which otherwise would enter the sphere's volume pass over the surface round to their target point.

17.4 Direction modifier

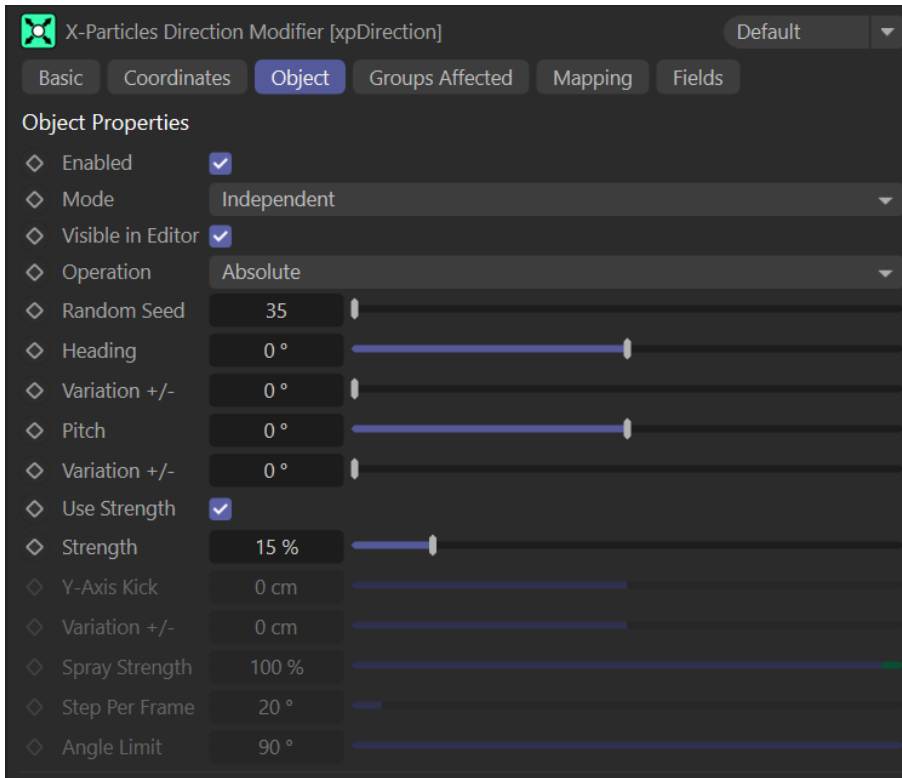


Figure 17.10. Direction modifier

What it does

The modifier changes the particle's direction of travel, using a variety of methods to do so. Particle speed is unaffected.

Why use it?

Many modifiers can change particle direction because it is inherent in what they do. The Direction modifier was originally designed to let you specify the exact direction a particle will move in. Since then several other modes have been added to enable various effects.

How to use it

Particle direction is stored internally as a vector which is not easy to interpret as a direction simply by looking at the value. For instance, what does a direction vector of $(-0.71, 0.71, 0.0)$ mean? It's not easy to visualise. (For information, this is a direction of 45 degrees along the negative X axis and positive Y-axis as in the screenshot in Figure 17.11, where the arrow points in the specified direction.)

i If you don't like the viewport display showing the new direction of travel, turn off the Visible in Editor switch. This doesn't affect modifier functionality, only the viewport appearance.

Many features in this modifier use a random number to assign direction or the strength applied to the direction change. For this reason it uses its own random number generator, the seed value for which is given in the Random Seed setting. You can change this if you don't like the results when random values are being used.

Heading and Pitch

Since direction vectors are difficult to visualise, the vectors are converted into angles, which are easier to understand. Only two angles - heading and pitch - are required for this, since the third angle - bank - does not affect particle direction. Depending on the modifier operation, you can alter one or both of these angles, or in two cases, neither of them is changed. Heading is the direction in the XZ plane, while Pitch is the direction in the YZ plane. The two combined together give the overall direction in 3D. You can add per-particle random variation to the angles used with the two Variation +/- settings.

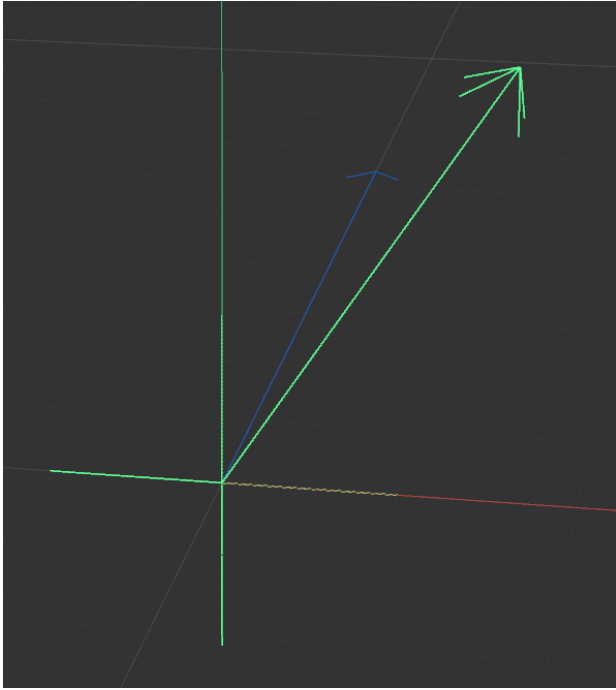


Figure 17.11. Representation of the specified direction in the viewport

⚠ The use of angles to describe direction is not perfect and occasionally anomalies can occur. The problem is that while a direction vector can be converted into angles, the result isn't always correct when converting angles back into a vector. For example, in the file [ch17_mods_direction_basic.c4d](#), try setting the heading to zero and the pitch to 180 degrees and watch what happens. Note that 179 and 181 degrees work more or less as expected but 180 does not! This is a consequence of trying to convert the specified angles into a vector. Let the scene run long enough and you will see that eventually 180 degrees pitch will work as expected, but with smaller angles the change of direction starts immediately. You can mitigate this with the Strength setting - see below for more details.

See the example file [ch17_mods_direction_basic.c4d](#). When run, the modifier appears to do nothing at all. This is because the particles are moving along the positive Z axis and the modifier does not change that. It is important to understand why. The particle direction in this scene is the direction in the 3D world, not relative to the emitter or some other object. Since by default in a new emitter particles will move along the +Z axis, the modifier treats that direction as a heading and pitch of zero degrees. But if you change the emitter so that the particles are initially emitted along the positive or negative X or Y axes, you will see that without changing any parameters the modifier will change the particle direction to +Z.

The default mode (in the Operation setting) is Absolute. This means that the specified direction is the actual 3D world direction, unaffected by the original direction of the particle. For example, if the original direction was along +Z (heading and pitch of zero) and the new heading was +90 degrees, the particles would change direction to move along the -X axis.

The opposite to this is Relative. In this case the new direction is relative to the current direction. With this mode, if the original direction was along +Z, a new heading of +90 degrees would add a number of degrees to the particle's heading each frame. In the example file, try setting the mode to relative and the Heading to 90 degrees. The result is what you would expect if the heading is constantly increased each frame - the particles move in a circle. Now try setting Heading back to zero and Pitch to 90 degrees. The result is...unanticipated. It is another of those anomalies referred to above consequent on the use of angles to describe direction.

Strength

i In earlier versions of X-Particles, this parameter was known as 'Acuteness of Turn' but is now named '**Strength**'.

To enable this parameter, first turn on the switch Use Strength. The strength setting controls how strongly the turn to the new direction is made. If it is set to zero, no change in direction will occur at all. With low values, the turn to the new direction is made in a gentle curve and takes a number of frames to reach the intended direction. As values increase, the curve gets tighter and the particle reaches the direction more quickly. If you turn off Use Strength the direction change is immediate, with no curve or delay before reaching the new direction.

In Absolute mode you can remove the problem discussed above which occurs when you set the pitch to 180 degrees. Either turn

Use Strength off or set the strength to a high value, such as 75% or higher. In Relative mode you cannot turn it off - you must a Strength setting of some value - but here, even setting it to 100% will not prevent the 90 degree of pitch problem mentioned above. Try it, and see what happens in relative mode, then try a setting of 1 to 5%. These low values will make the pitch work correctly, but you will still see anomalies depending on the pitch angle and strength setting. For example, in relative mode try a pitch of 270 degrees and varying strength values from low to high.

Spray mode

This mode only has one setting. Spray Strength. By default it is set to 100% but actually better results are achieved with low numbers such as 2 to 5%. What it does is cause the particles to spray out from their original direction. See the file [ch17_mods_direction_spray.c4d](#) for a very simple example. Best results are obtained with a very low spray strength, an emitter source which is very small, and where the particles are originally unidirectional. Try changing the emitter in the example file to spherical and you can't see any effect of this mode.

Circular

As the name implies, this causes the particles to move in a circle. You can control the size with the Heading and Strength parameters. Actually, in this mode you only need one; the other is redundant, in that you can achieve the same results with different combinations of the two settings, but having both does at least allow you keyframe one of them and keep the other one the same.

We've already seen that you can make particles move in a circle with the Relative mode but this mode has two advantages. Firstly, the circle is constructed in the emitter's (not the modifier's!) YZ plane. If you want to change the plane, simply rotate the emitter. See the example file [ch17_mods_direction_circular.c4d](#). If you play it, the particle describes a perfect circle in the YZ plane. Rotate the emitter in any plane(s) and you still get a perfect circle but its plane is changed. Now reload the file and change Operation to Relative, leaving everything else unchanged. Rotate the emitter in one or more planes and play it; the chances are you will not get a circle at all.

The other advantage is parameter Y-Axis Kick. This alters the particle direction in the world Y axis, which lets you show spirals such as this:



Figure 17.12. Circular mode with Y-Axis Kick applied

The file [cb17_mods_direction_circular_spring.c4d](#) shows how this is done.

! The limitation of this feature is that it only works in the Y axis of the 3D world, not that of the emitter or modifier.

Jitter

In this mode the particle direction jitters around the heading and/or pitch values. In other words, if the heading is 10 degrees, the direction will change each frame by up to plus or minus 10 degrees of heading. This can produce some quite pleasing organic shapes; see the file [cb17_mods_direction_jitter.c4d](#) for an example. There are no additional parameters with this mode, just the usual heading, pitch and strength settings,.

Ring

This mode will force the particles to adopt a ring, or torus, shape. As with Spray mode, which it resembles, best results are obtained with a small emitter and particles all moving in the same direction. See the simple example file [cb17_mods_direction_ring.c4d](#), which uses an emitter in Pulse mode to produce results like this:

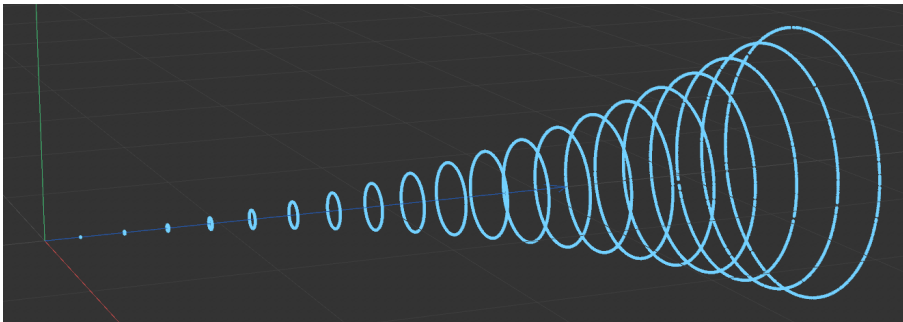


Figure 17.13. Ring mode

i In this file, subframe emission has been disabled in the emitter to achieve more aesthetically-pleasing results.

The way this works is that the particles deviate from their original direction at a random angle from their original path. For example, if the particles were originally moving along the X axis, they would change direction so that they also move along Y and Z as well. Although each particle deviates at a different angle, they all do so at the same rate, which gives the ring effect. The speed at which the ring grows is controlled by the Step Per Frame setting.

The final setting for this mode is Angle Limit. This governs how big the angle of deviation can be from the original direction. If this is set to 30 degrees, the particles will not deviate more than that. This doesn't stop the ring expanding, just stops any increase in the angle of deviation. In Figure 17.14, the screenshot shows the blue emitter has an angle limit of 15 degrees, but the orange emitter has a limit of 90 degrees (example file [cb17_mods_direction_ring_limit.c4d](#)):

Use Modifier Rotation

This final direction mode lets you rotate the modifier and use its Z axis as the new direction. It works exactly the same as Absolute mode except that you rotate the modifier itself rather than entering heading and pitch values. You can still add variation to the heading and pitch, and change the strength, but the heading and pitch values are taken from the modifier rotation and cannot be entered into the modifier's interface.

! Altering the modifier bank rotation has no effect since as previously mentioned, bank does not affect direction.

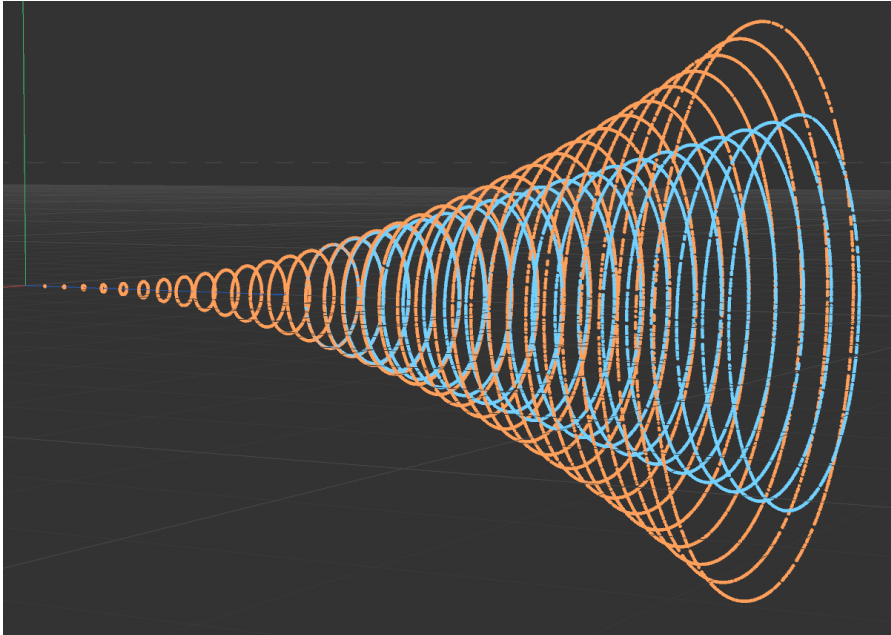


Figure 17.14. Ring mode with different Angle Limit values

17.5 Drag modifier

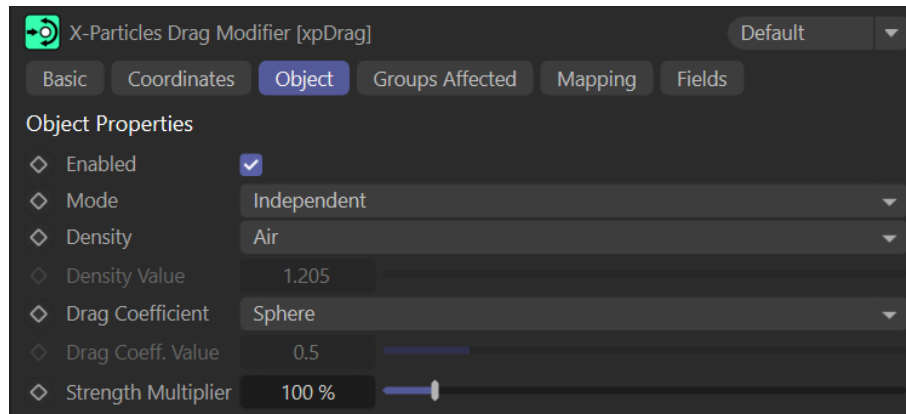


Figure 17.15. Drag modifier

What it does

The Drag modifier slows particles down; it can't speed them up. It does this by simulating what would happen if the particles were moving through a gas or liquid. In a vacuum there is no resistance to a moving object but with any other medium there will be resistance to movement and the object will slow down. The amount of resistance, and therefore how quickly the object slows, depends on three things:

- the density of the medium - basically, how 'thick' the medium is, so that for example, air has lower density than water, which has a lower density than treacle
- the shape of the object - some shapes are more affected by the resistance of the medium than others, so a sphere offers less resistance than a cube; this is measured with the 'drag coefficient' of the object
- the size of the object - for objects of identical shape, larger objects are more affected by resistance than smaller ones

The Drag modifier lets you choose from a variety of media with differing densities, or you can set your own custom density value. Although it doesn't assess the shape of the particle to assign a drag coefficient to it, it offers a variety of shapes with known coefficients and you can choose which one is the best approximation to your particle shape or generated object.

Why use it?

Simply put, to simulate the effect of drag, such as air resistance, on a moving particle or its generated object.

How to use it

This is a simple modifier to use. First, choose the density of the medium surrounding the particles from the **'Density'** menu. The default is **'Air'** but you can choose **'Vacuum'** (in which case the modifier has no effect) or a wide variety of other gases or liquids. When you choose one, the relative density of that medium is shown in the **'Density Value'** setting, although you cannot change it. This may be useful if you want to set a custom density since the preset values will give you an idea of what value you might need. To set your own density value, select **'Custom'** from the menu.

Next, choose the **'Drag Coefficient'** you want from that menu. Ideally you would choose whatever shape best approximates to the shape of the particle or its associated object. Again, you can see the drag coefficient value for the shape you select in the **'Drag Coeff. Value'** setting and can set your own value by choosing **'Custom'** from the menu. There is one other entry which can be useful, **'xpShatter Fragments'**. This is a preset which is a good starting point for the irregularly-shaped objects generated by the X-Particles Shatter object.

Finally, you can alter the strength of the effect up or down using the **'Strength Multiplier'** setting. This is not a real-world setting, it's there simply to allow fine-tuning of the effect.

With regard to the effect of size, the particle radius is used for this. Larger particles are more affected by drag than smaller ones.

That's really all there is to it. As an example, see the file [cb17_mods_drag_basic.c4d](#). This shows a shot of particles with varying radii, and you can see that the larger particles are more affected than the small one. What this file also shows is how the Drag modifier plays with other motion modifiers. Turn on the Gravity modifier in the object manager and you see that the drag doesn't have much effect, because the gravity acts to overcome the drag. But if you change the density to **'Propane (Liquefied)'** you see a much greater effect in that small particles are pulled down by gravity while the largest ones hardly move at all. You can see a similar result if you turn off the Gravity modifier and turn on the Attractor modifier.