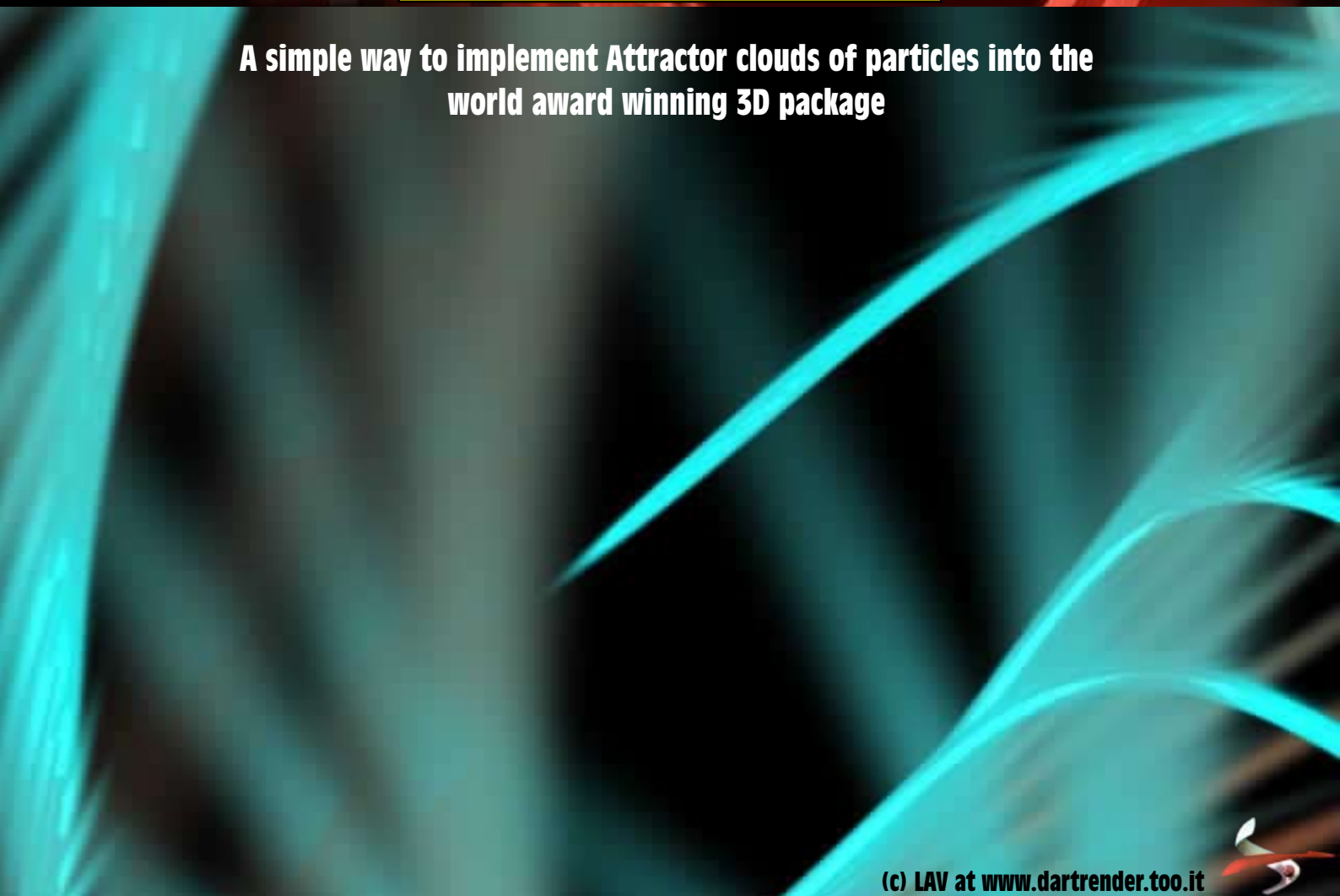
Aurora's Tim Dunn plugin

Attractors & Lightwave3D

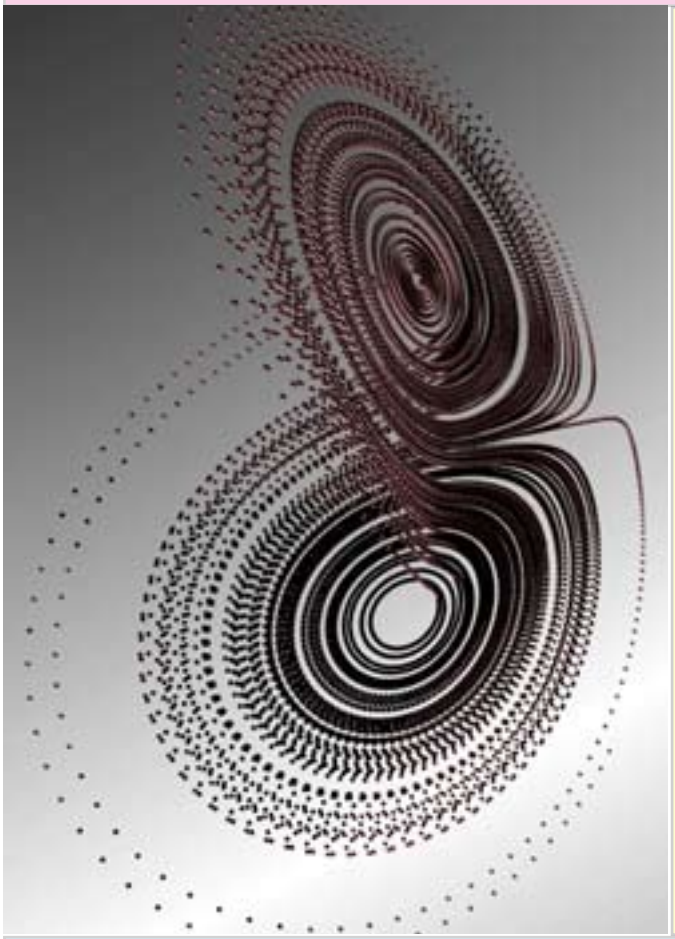
A simple way to implement Attractor clouds of particles into the world award winning 3D package



Attractors by a simple math point of view

Attractors are maths objects. In effect, you can't see them. In the real world, I mean.
Nevertheless they are recognized as 'engines' leading on several real world things.
Basically an *attractor* is a formula and a recursive algorithm that generates, for each recursion, a point in an X,Y,Z cohordinate of the 3d space. But I'm not an expert, so let me dig in the net.

In dynamical systems, an attractor is a set to which the system evolves after a long enough time. For the set to be an attractor, trajectories that get close enough to the attractor must remain close even if slightly disturbed. Geometrically, an attractor can be a point, a curve, a manifold, or even a complicated set with fractal structures known as a strange attractor. Describing the attractors of chaotic dynamical systems has been one of the achievements of chaos theory. (*Wikipedia, attractor*)
One of the most simplest attractor is known as Lorentz .



The lorenz attractor was first studied by Ed N. Lorenz, a meterologist, around 1963. It was derived from a simplified model of convection in the earths atmosphere. It also arises naturally in models of lasers and dynamos. The system is most commonly expressed as 3 coupled non-linear differential equations.

$$\frac{dx}{dt} = a(y - x)$$

$$\frac{dy}{dt} = x(b - z) - y$$

$$\frac{dz}{dt} = xy - c\,z$$

The series does not form limit cycles nor does it ever reach a steady state. Instead it is an example of deterministic chaos. As with other chaotic systems the Lorenz system is sensitive to the initial conditions, two initial states no matter how close will diverge, usually sooner rather than later. (*http://local.wasp.uwa.edu.au/~pbourke/fractals/lorenz/ where left image is hosted*).

The so called *rossler* system is credited to Otto Rossler and arose from work in chemical kinetics. The system is described with 3 coupled non-linear differential equations

$$\frac{dx}{dt} = -y - z$$

$$\frac{dy}{dt} = x + a\,y$$

$$\frac{dz}{dt} = b + z(x - c)$$

where $a = 0.2$, $b = 0.2$, $c = 5.7$

The series does not form limit cycles nor does it ever reach a steady state. Instead it is an example of deterministic chaos. As with other chaotic systems the Rossler system is sensitive to the initial conditions, two initial states no matter how close will diverge, usually sooner rather than later.

While the equations look simple enough they lead to wonderful trajectories, some examples of which are illustrated in this page

The following is a short piece of simple C code to illustrate how one might create the attractors shown here.

```
  double h = 0.05; /* or smaller */
  double a = 0.2;
  double b = 0.2;
  double c = 5.7;
  XYZ p,plast = {0.1,0,0};

  for (i=0;i<1000000;i++) {
    p.x = plast.x + h * (- plast.y - plast.z);
    p.y = plast.y + h * (plast.x + a * plast.y);
    p.z = plast.z + h * (b + plast.z * (plast.x
- c));
    if (i > 100)
      Draw the point p
    plast = p
  }
```

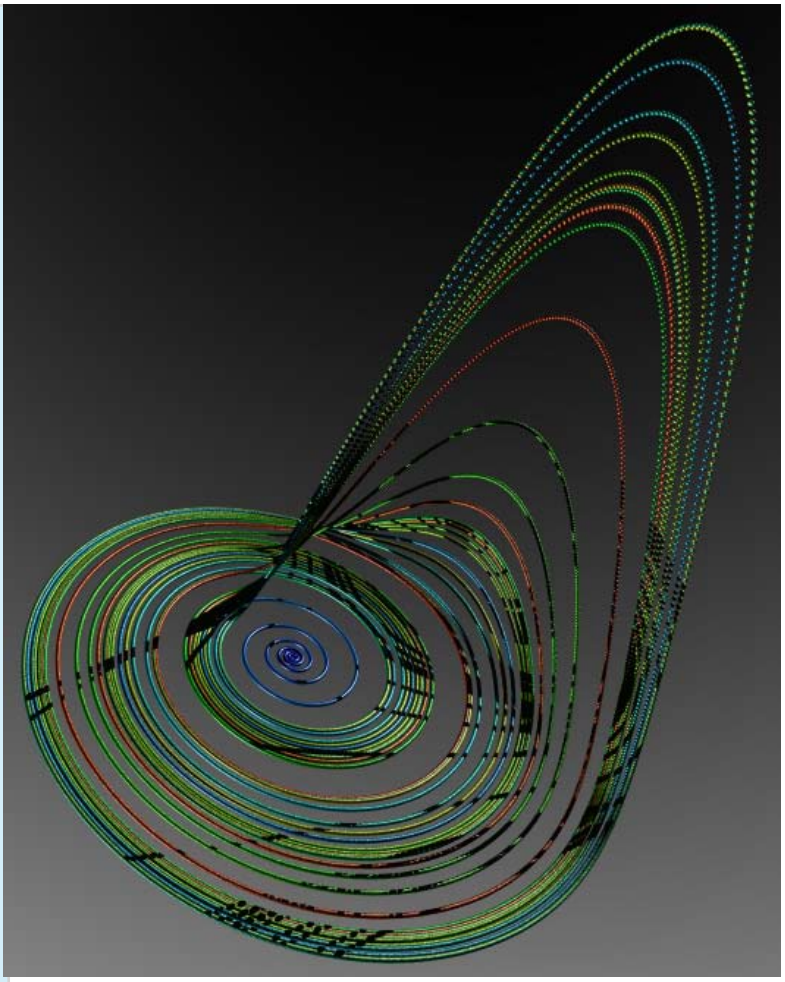
(*http://local.wasp.uwa.edu.au/~pbourke/fractals/rossler/ + right upper image*)

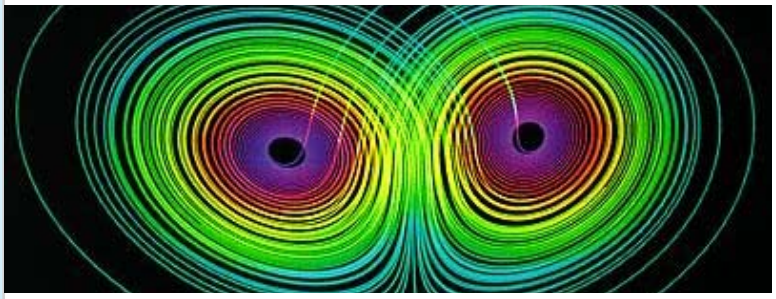
If you give a sight at the code above, you can imagine that each point has an extra value, other then x,y,z. The i(ndex) of the loop could be the *time* you can draw the color of point with.

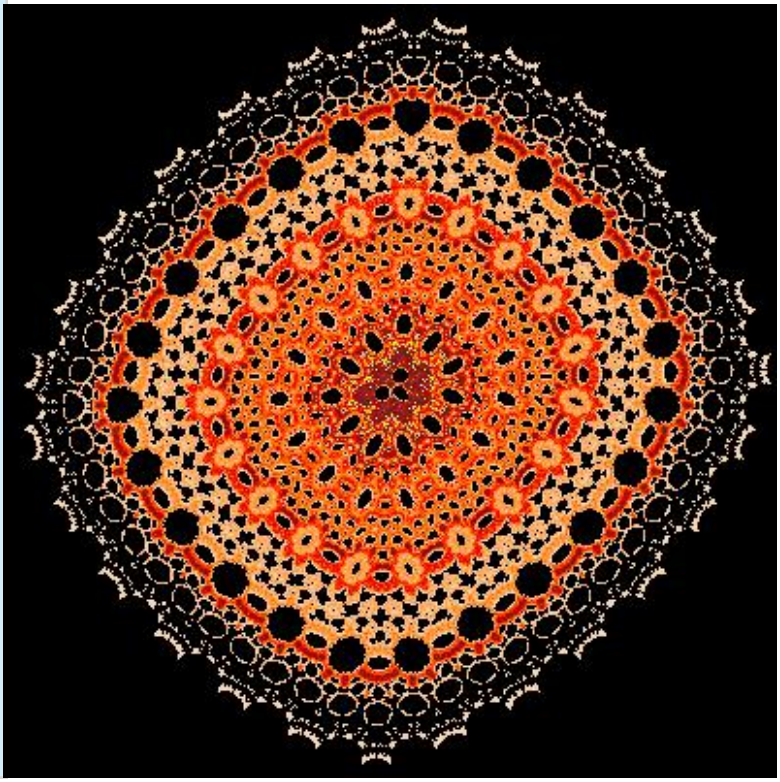
This is a brief and small introduction to the attractor world: by our point of view we just need to know that this kind of objects can easily obtained by the machine we use to make 3d. Lightwave has an integrated system that let us write these small pieces of code to make attractors interact with other Lightwave instruments, such as points, polygons, particles and colors.

The target of this tutorial is to show how we can use Aurora - Tim Dunn code to do this extra exploration in mathematical and abstract spaces.

(on the right: lorenz and hopalong attractors)







Simple project

Now let's go to Lightwave and attractors.

First of all you have to download *Aurora's attractors* plugin from Tim site www.auroragrafx.com (in that page you can find a global explanation of the plugin itself).

.

Then you need to load it into Lightwave. You know: you put the *.p file into your plugs directory, launch Layout, use ALT+F10 key combination, load plugin, browse to directory and load it.

Now you have the code into object properties -> custom object panel. Actually the plugin acts only as a sort of modifier when connected to an object. You can use a NULL object in order to let the attractor appears.

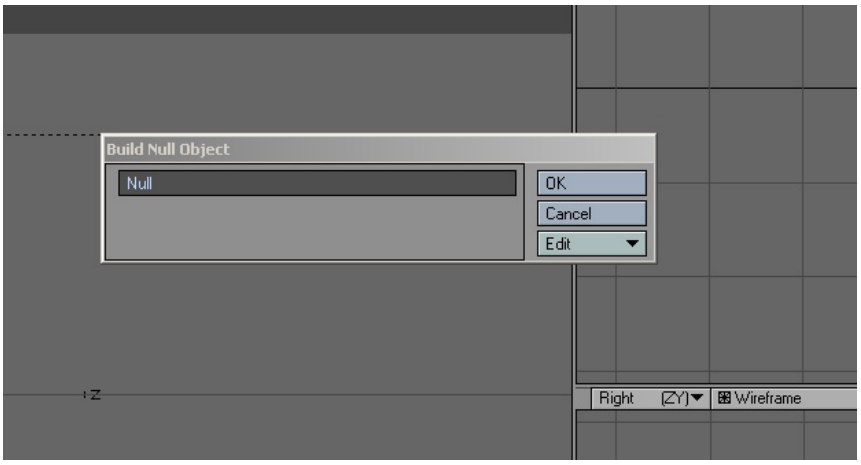


fig:1 - Add null

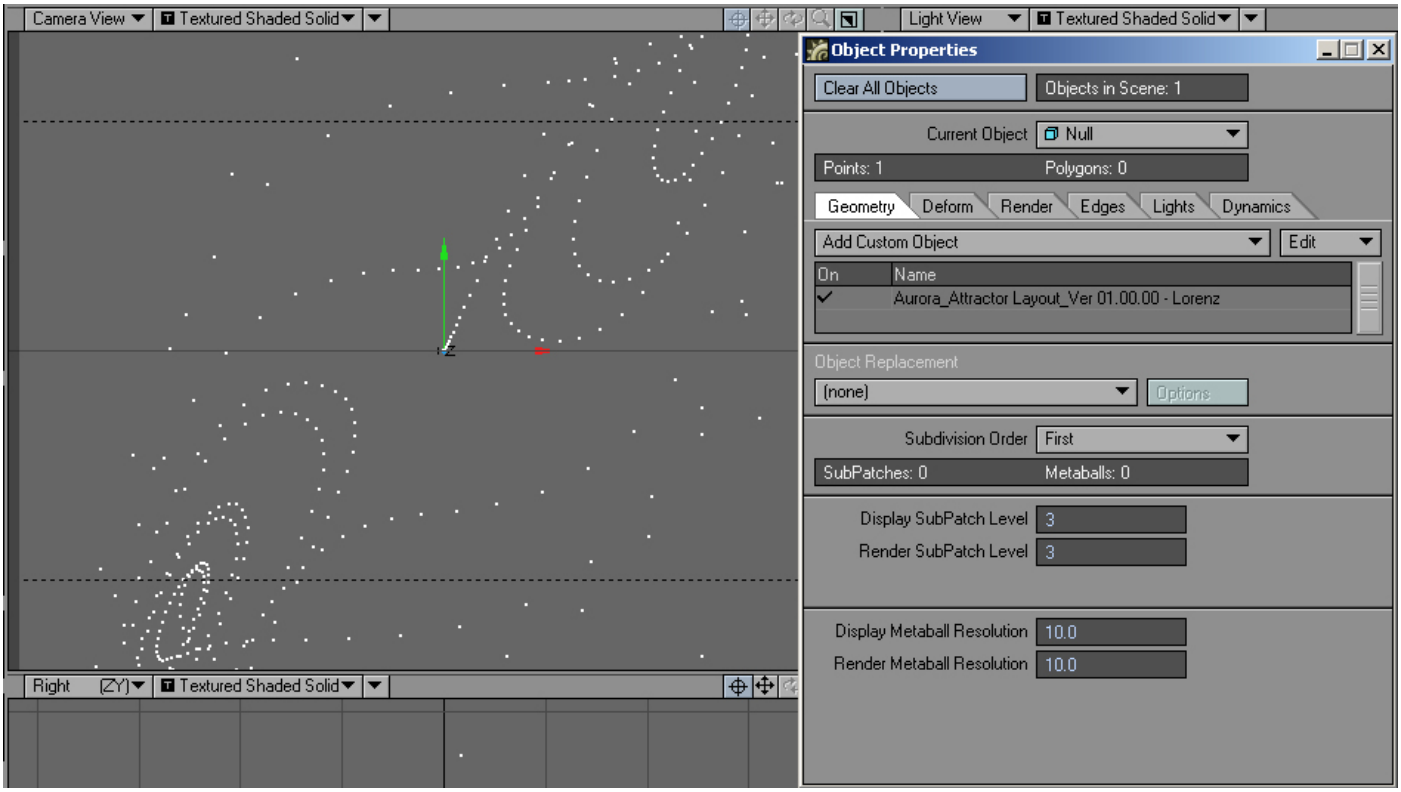
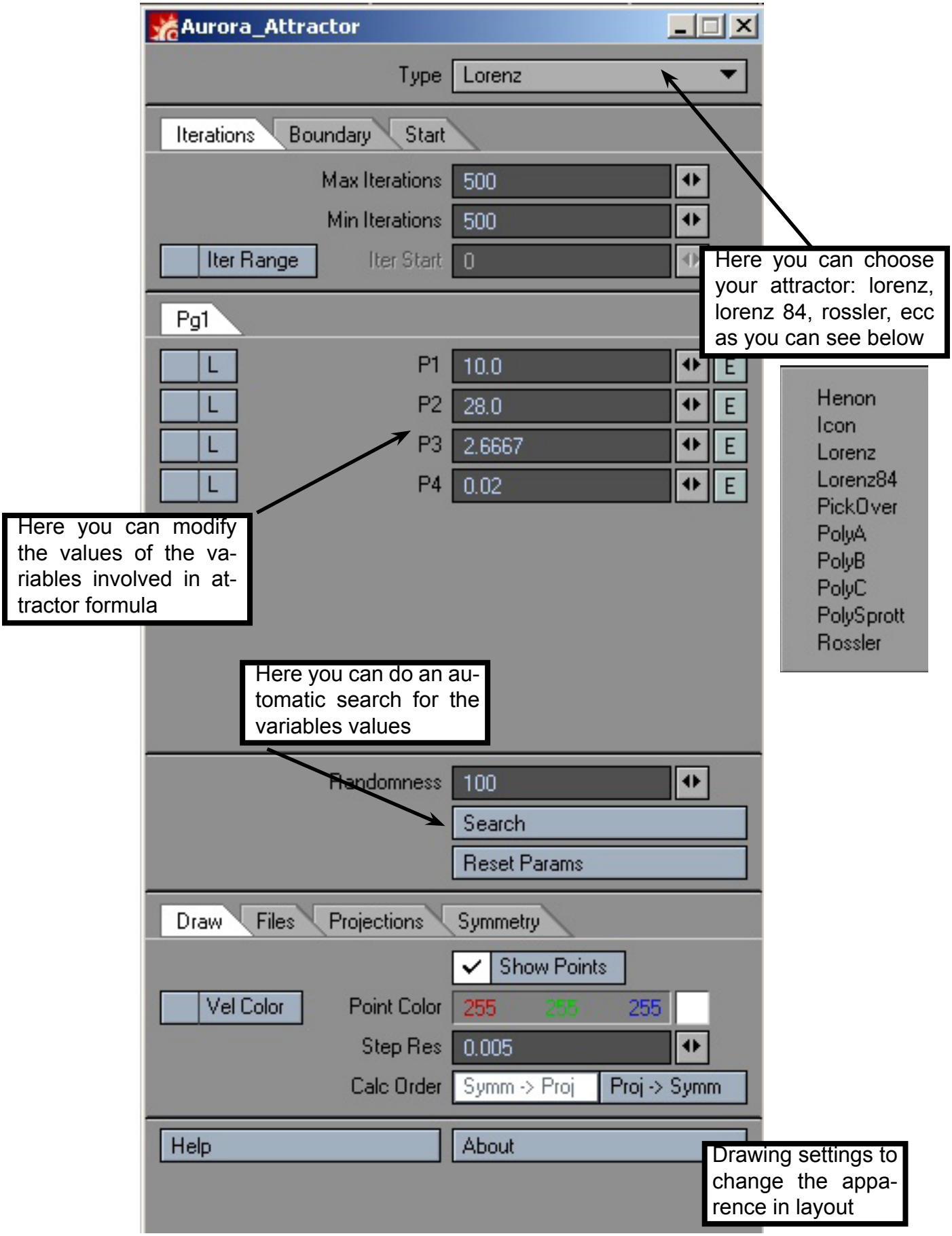


Fig. 2: 'p' to object properties > You add Aurora Attractor in Geometry> Custom object; immediately you have the default attractor drawn on the quad view.

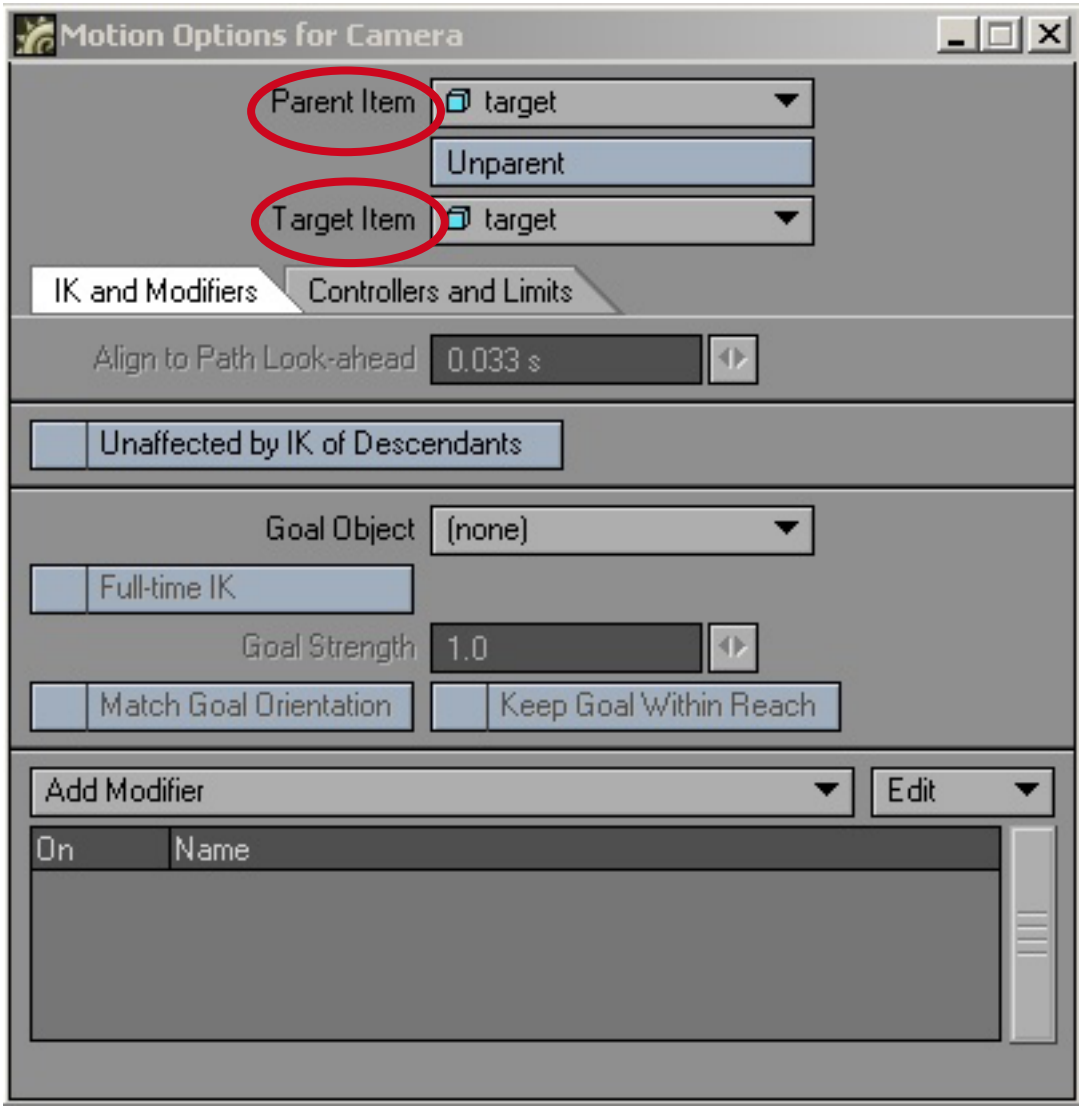


Double clicking on the name of the plugin causes the opening of a panel that's the core of the problem.
For now we are not worryed about the subpanels (Boundary, start, Files, Projection and Symmetry). Let's play with the main options and see what happens.

Now the best is adding another null object to use as target for our camera. Aiming camera is a critical step in rendering attractors.

Then:
1) add null and name it such as ‘target’
2) select camera,
3) press ‘m’ to open motion panel,
4) select ‘target’ as Parent item and Target item.

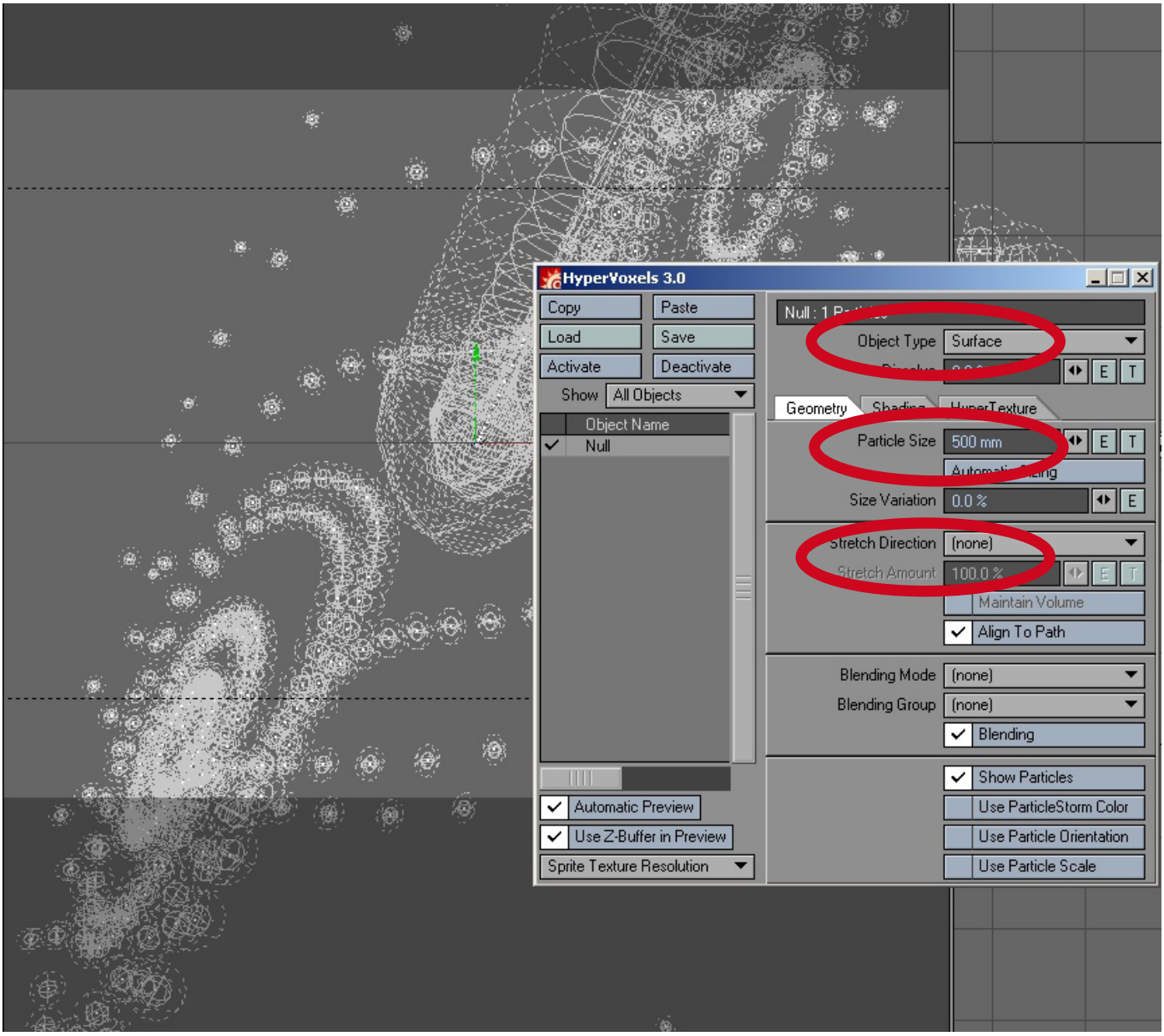
Now we have camera locked to the new null. You can move null and camera itself will move to follow the null. But you can move camera indipendentely to tune other settingd, i.e. the point of view.



Ok. Now we have a simple but complete scene with NO polygon! The setting for a tipical attractor scene is the simplest you can set in Lightwave: a pair of nulls, a camera, a normal set of lights and... stop!

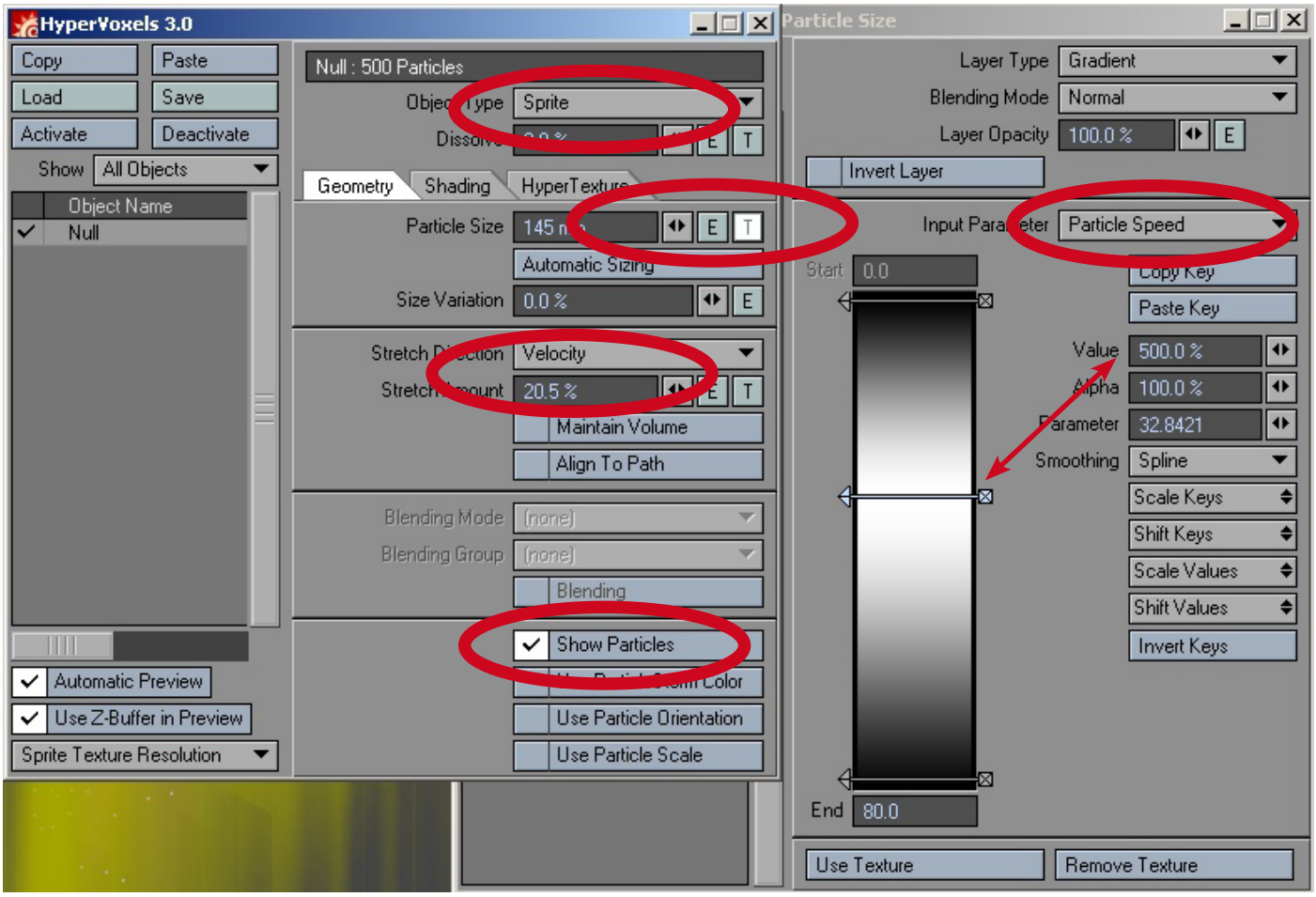
Now the things become more familiar for a normal Lightwave user: we have a cloud of particles as we obtained them transforming a null in an emitter. We have just to set the particles visibility adding the Hyperwoxels modifier. We transform aset of attractor points into a volumetric object.

So open Hyperwoxels panel (if you didn’t change the default setting, you can find it in ‘window’ sub-menu) and check your attractor null. Check ‘Show particles’ too and, hoplà, you have this screen.

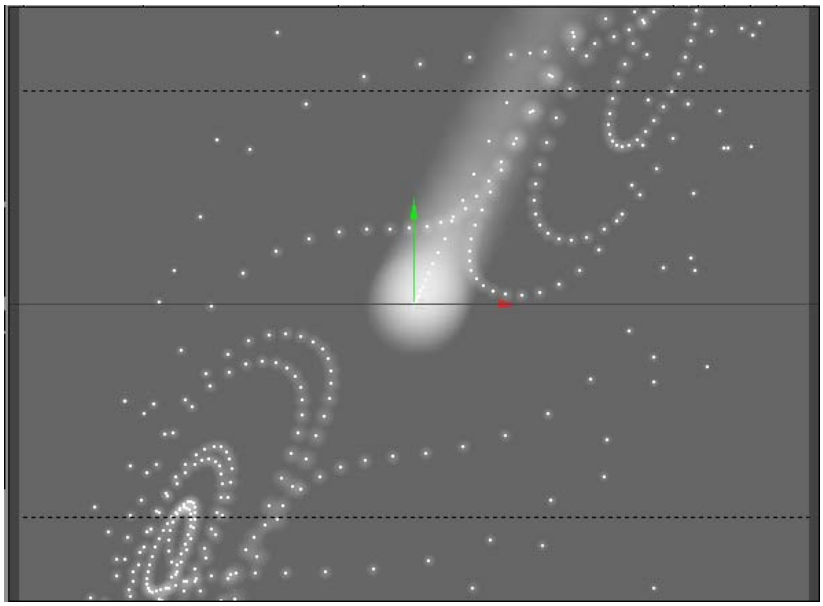


Now we are surely anxious to give colors and shapes to our particles. So we need to work with all the options of the Hypervoxels panels.
Let me be your guide in this exercise:
1) set ‘Object Type’ as Sprite. The Sprite mode is fast and fit with attractors
2) open Viper and let it enjoy you with its almost immediate feedback
3) Change the value of Particle size. This value determines what is the ‘starting’ size of particle. you can change and shape the particle
4) Changing the stretch value - With Direction = none, stretch is deactivated.

As you know the stretching valie is used to ... stretch particles along X,Y,Z or Velocity. In normal particle generated by a normal emitter, this value corresponds to the normal movements in the space/time. In the attractor you are working with, Velocity is a strange and exoteric value that rapresents... hmmm... let Aurora speaks:”For shading your particles I would recommend starting with simple sprites. Due to the funky mathematical nature of strange attractors you can calculate the velocity, as it were, as the numerical solutions fly around the basins of attraction. I have done this in the plugin and provided you access to this information by setting a color gradient and setting the input parameter to ‘Particle Speed’. Currently the velocity is non-normalized for the various attractors. What this means to you is you’ll want to play with adjusting the ‘End’ value of the gradient and scaling your gradient till you get the desired result.” (www.auroragrafx.com)

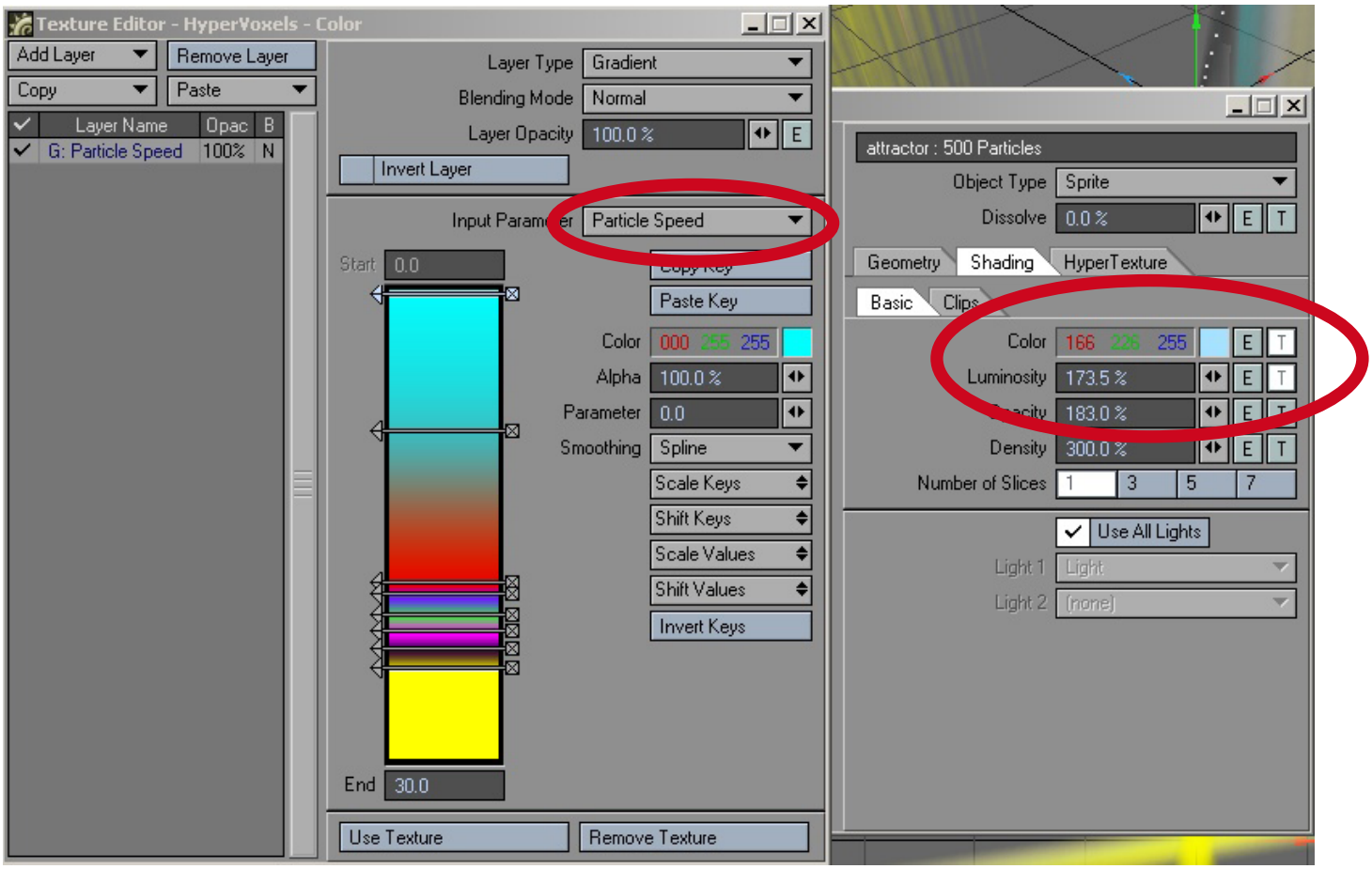


Click on T(exture) of Particle size and set the layer as a gradient. Let the gradient refer to Particle Speed or to Distance by Object (you can the choose the object as null-attractor itself or null-target, or whatever you want). I don't know exactly what Particle speed means, but I found it very useful. Playing with these values, with 'show particles', checked you can have an immediate feedback into OpenGL quad views.

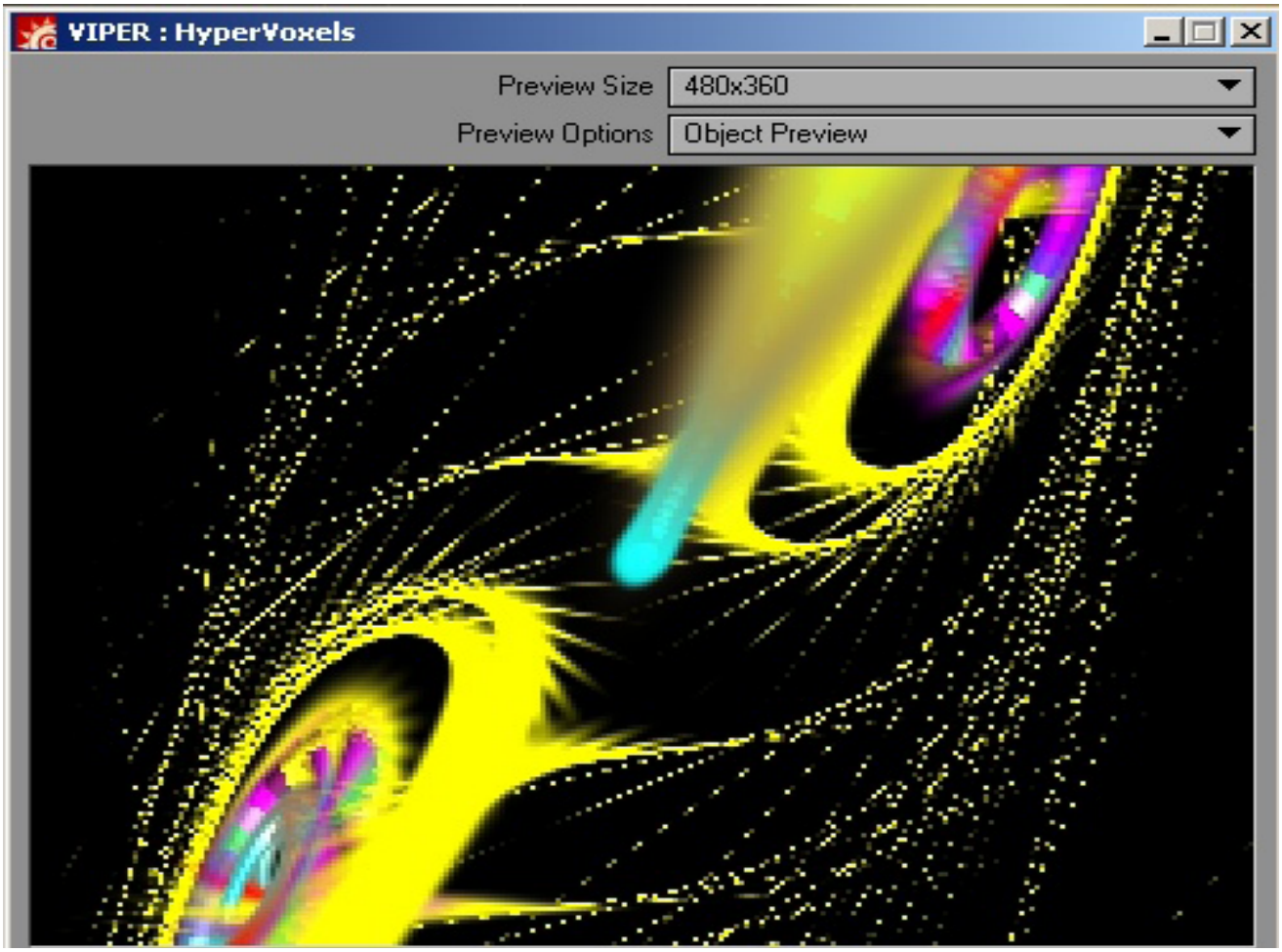


When you are fit with the particles size, go on and work with 'Stretch' values. You can set Stretch direction to Particle Speed, the stretch value as you like (up to 300-500%) and then open the Texture panel (Stretch amount-> T button). To go on with this simple exercise, we can just copy the Particle Size->Texture layer (the above gradient) and paste into Stretch amount layer panel. You can change the layer input as you want. I usually work on Particle Speed.

You select the main Shading panel in HW window and set the main color. Then you click on the ususal T button and set the layer as Gradient. As usual: chosse Gradient, Particle Speed, and set the colors steps as you like.



This is what you can see in Viper window



The colors you added to attractor cloud let you have an immediate (Viper) feedback of the changing object.

For now I stop here. In the next tutorial, I hope write it in a short time, you can find a lot of exercises: turning particles in surfaces, texturing surfaces, mixing real objects with attractors and other