Particle Shading

This page provides a tutorial on using the Phoenix Particle Texture to shade a particle simulation with Chaos Phoenix in 3ds Max.

Overview

This is an Intermediate Level tutorial. Even though no previous knowledge of Phoenix is required to follow along, re-purposing the setup shown here to another shot may require a deeper understanding of the host platform's tools, and some modifications of the simulation settings.

Requires **Phoenix 3.11.00 Official Release** and **V-Ray Next Official Release** for 3ds Max 2015 at least. You can download official Phoenix and V-Ray from https://download.chaos.com. If you notice a major difference between the results shown here and the behavior of your setup, please reach us using the Support Form.

The instructions on this page guide you through the process of using the Phoenix Particle Texture to shade a particle simulation with Phoenix in 3ds Max.

The **Download** button below provides you with an archive containing the scene file.

Download Project Files

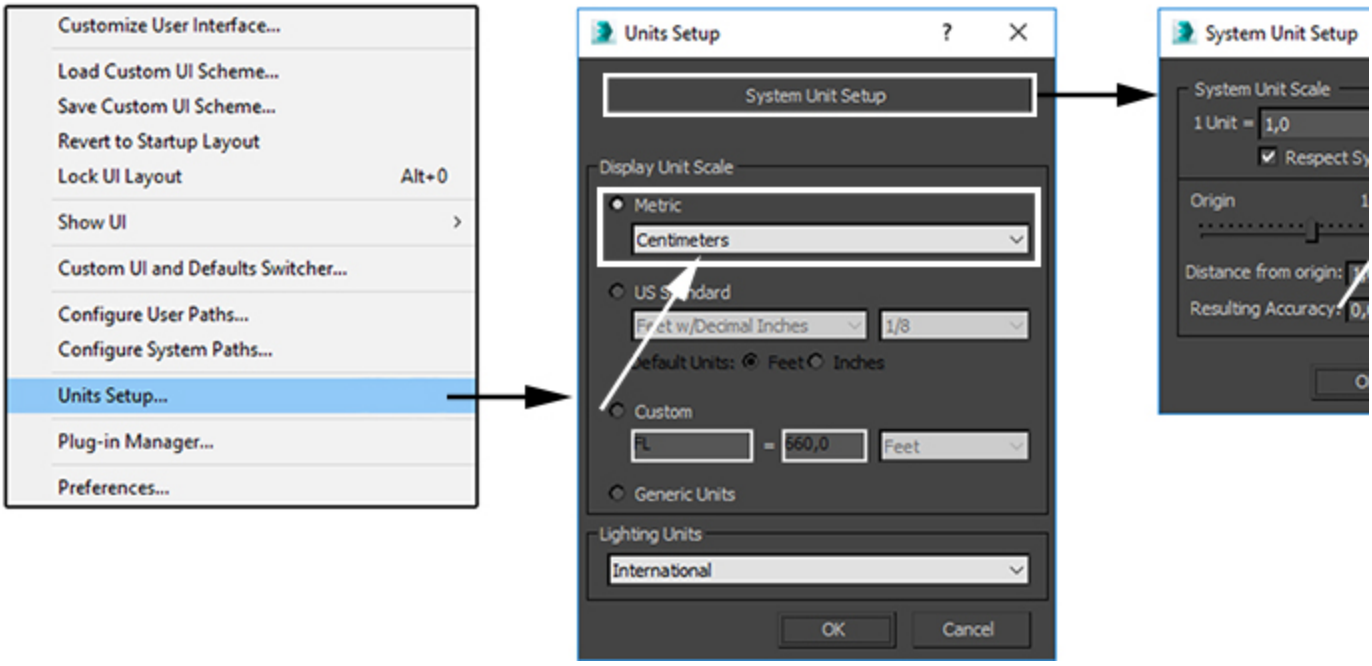
Units Setup

Scale is crucial for the behavior of any simulation. The real-world **size of the Simulator** in **units** is important for the simulation dynamics. Large-scale simulations appear to move more slowly, while mid-to-small scale simulations have lots of vigorous movement. When you create your Simulator, you must check the **Grid** rollout where the real-world extents of the Simulator are shown. If the size of the Simulator in the scene cannot be changed, you can cheat the solver into working as if the scale is larger or smaller by changing the **Scene Scale** option in the **Grid** rollout.

The Phoenix solver is not affected by how you choose to view the Display Unit Scale - it is just a matter of convenience.

Go to **Customize Units Setup** and set Display Unit Scale to **Metric Centimeters**.

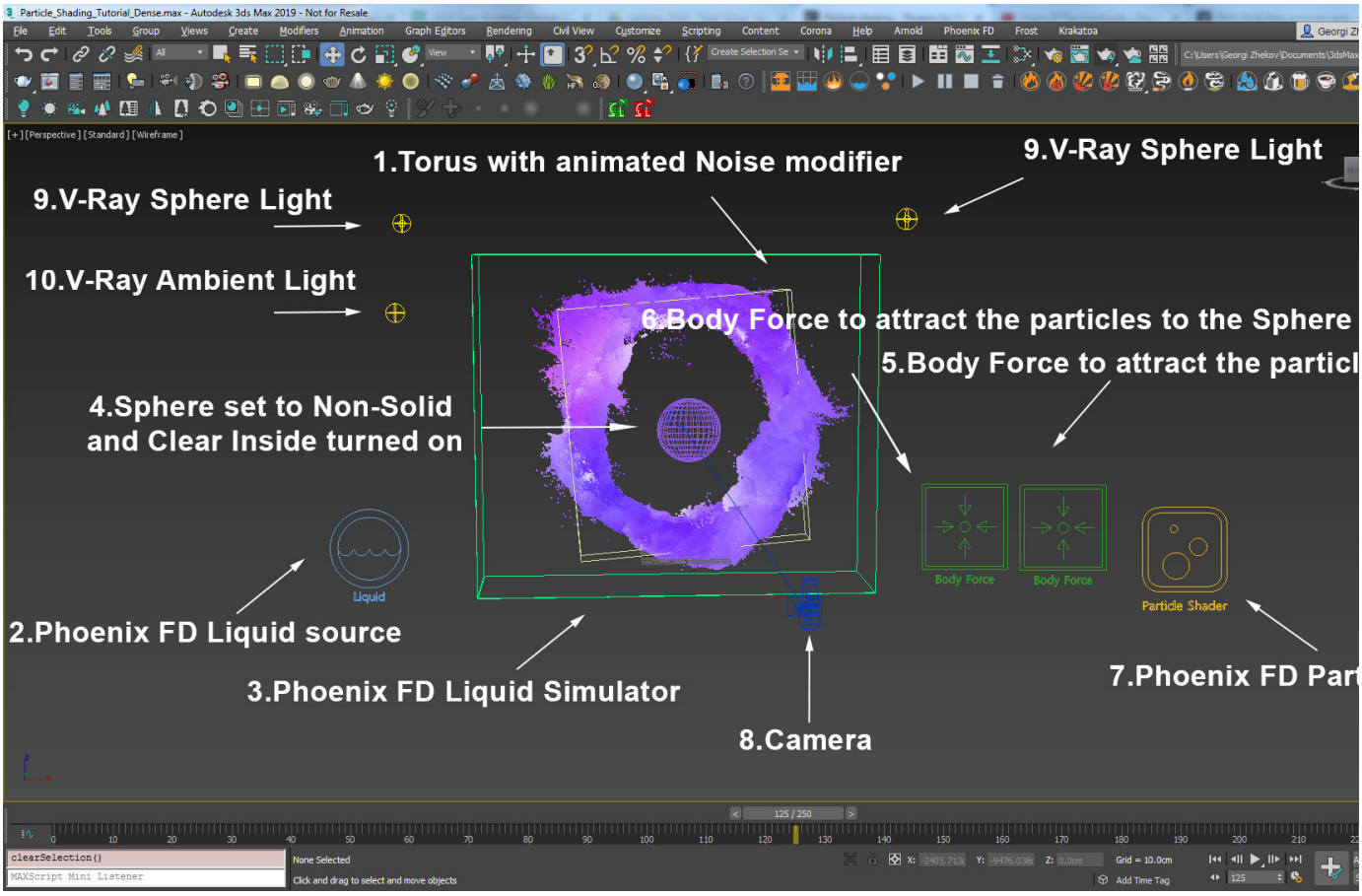
Also, set the **System Units** such that **1 Unit** equals **1 Centimeter**.



Scene Layout

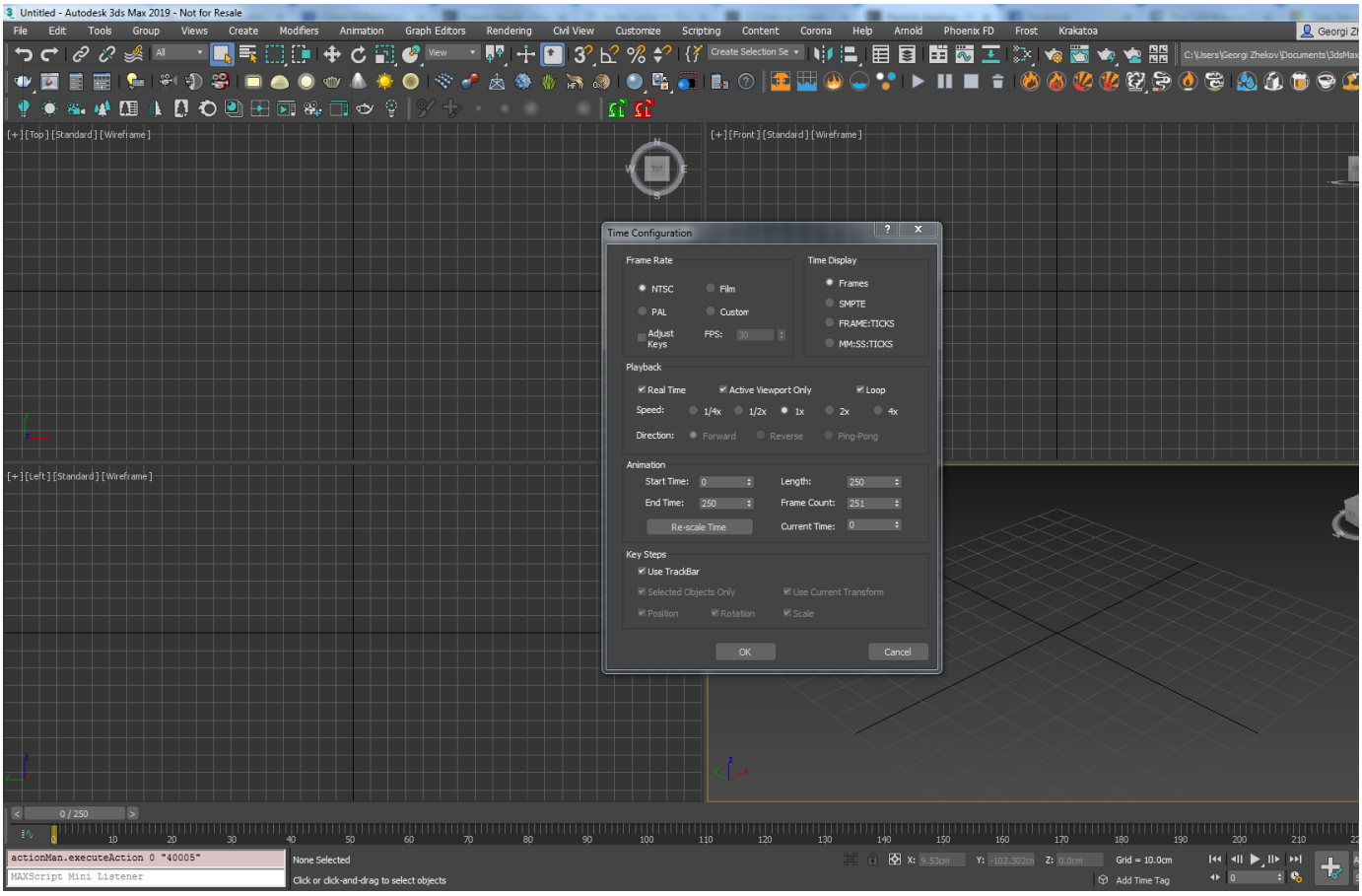
The final scene consists of the following elements:

1. A **Torus** used as the **source geometry** for Liquid particles. An animated **Noise modifier** is applied to the torus to break-up the motion of the liquid and produce interesting swirls.
2. A **Phoenix Liquid Source** with the **Torus** in its **Emit ter Nodes** list. The Source is in **Volume Brush** mode and is animated from 100% at frame 13 to 0% at frame 14.
3. A **Phoenix Liquid Simulator** with some tweaks in the **Grid and Dynamics** rollouts.
4. A **Sphere** set to **Non-solid** object from the Phoenix Properties and with **Clear Inside** turned on in order to delete all the particles that get inside of its volume.
5. A **Phoenix Body Force** used to attract the liquid particles to the Torus. The Strength is animated from **300** at frame **140** to **100** at frame **145**.
6. A **Phoenix Body Force** to attract the liquid particles to the Sphere. The Strength is animated from **0** at frame **140** to **800** at frame **145**.
7. A **Phoenix Particle Shader** used to render the liquid particles.
8. **A Standard Physical Camera** for rendering.
9. A **V-Ray Sphere Lights** for lighting the scene.
10. A **V-Ray Ambient Light for lighting the scene**.



Scene Setup

Set the **Time Configuration Animation Length** to 250 so that the Timeline goes from 0 to 250.

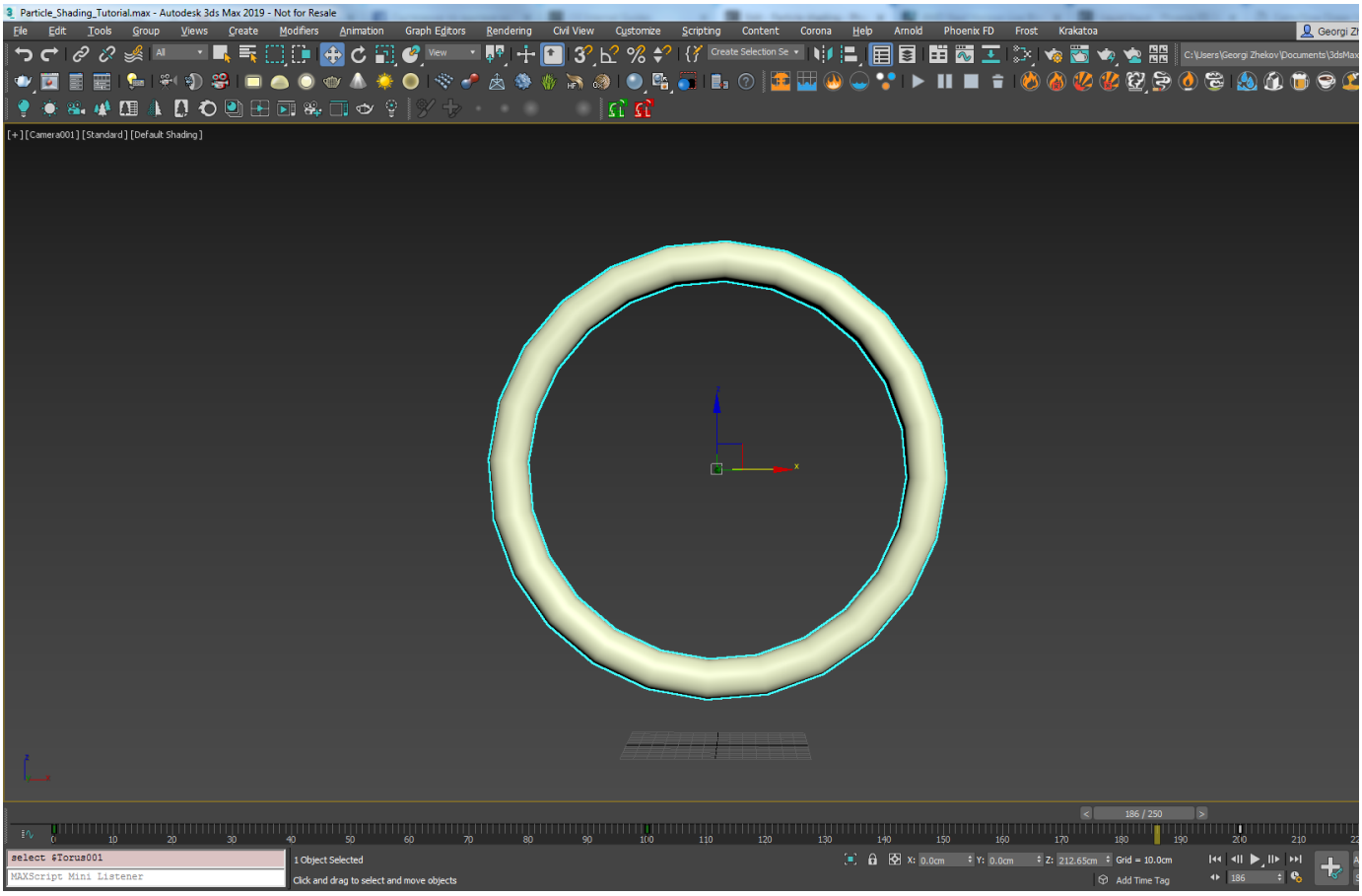


Create a **Standard Primitives Torus**. The torus will be used as the emission for the liquid particles.

Set its **Radius 1** to **162** and set the **Radius 2** to **15**.

Rotate the Torus **90** degrees on the **Y axis**.

Animate the **Y rotation** from **0** at frame **0** to **720** at frame **200** and set the keyframes to **linear interpolation** so that the animation will be constant through the whole range.



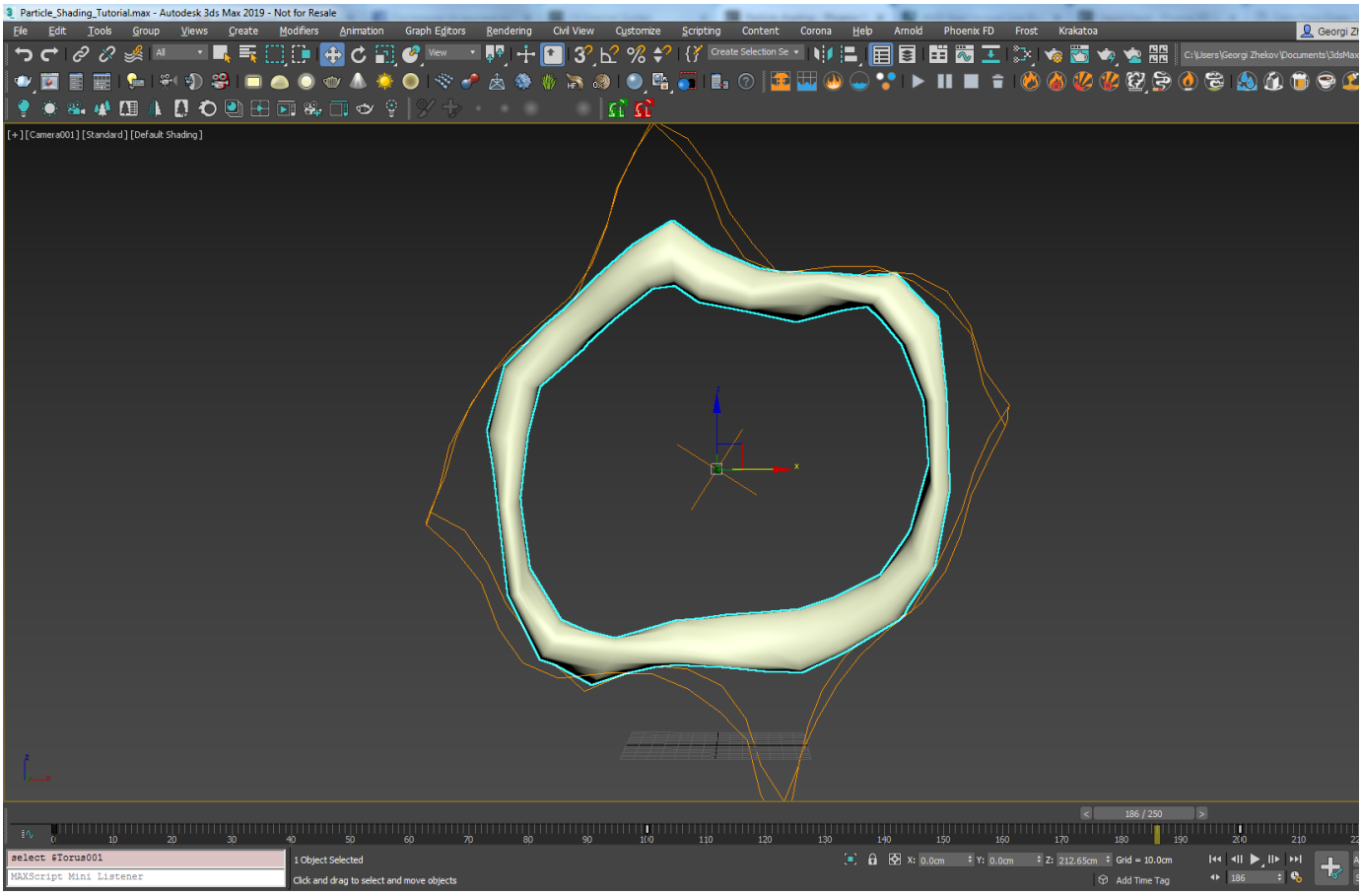
Apply a **Noise modifier** to the torus to give it irregular shape.

Set the **Scale** to **136**. Turn **Fractal** on.

For the **Strength** set for **X:85 Y:35 Z:28**.

Turn on **Animate Noise** and animate the **Phase** from **0** at frame **0** to **100** at frame **100**.

Using open geometry or geometry with no thickness can give you unpredictable simulation results. Making sure that your geometry is clean is crucial for a smooth workflow. Phoenix (and many simulation packages in general) use a volumetric representation of the emission geometry for the simulation. The process of creating this volumetric representation is called *voxelization*. The algorithms responsible for *voxelizing* the geometry can fail when using open (with holes) or planar (no thickness) geometry.

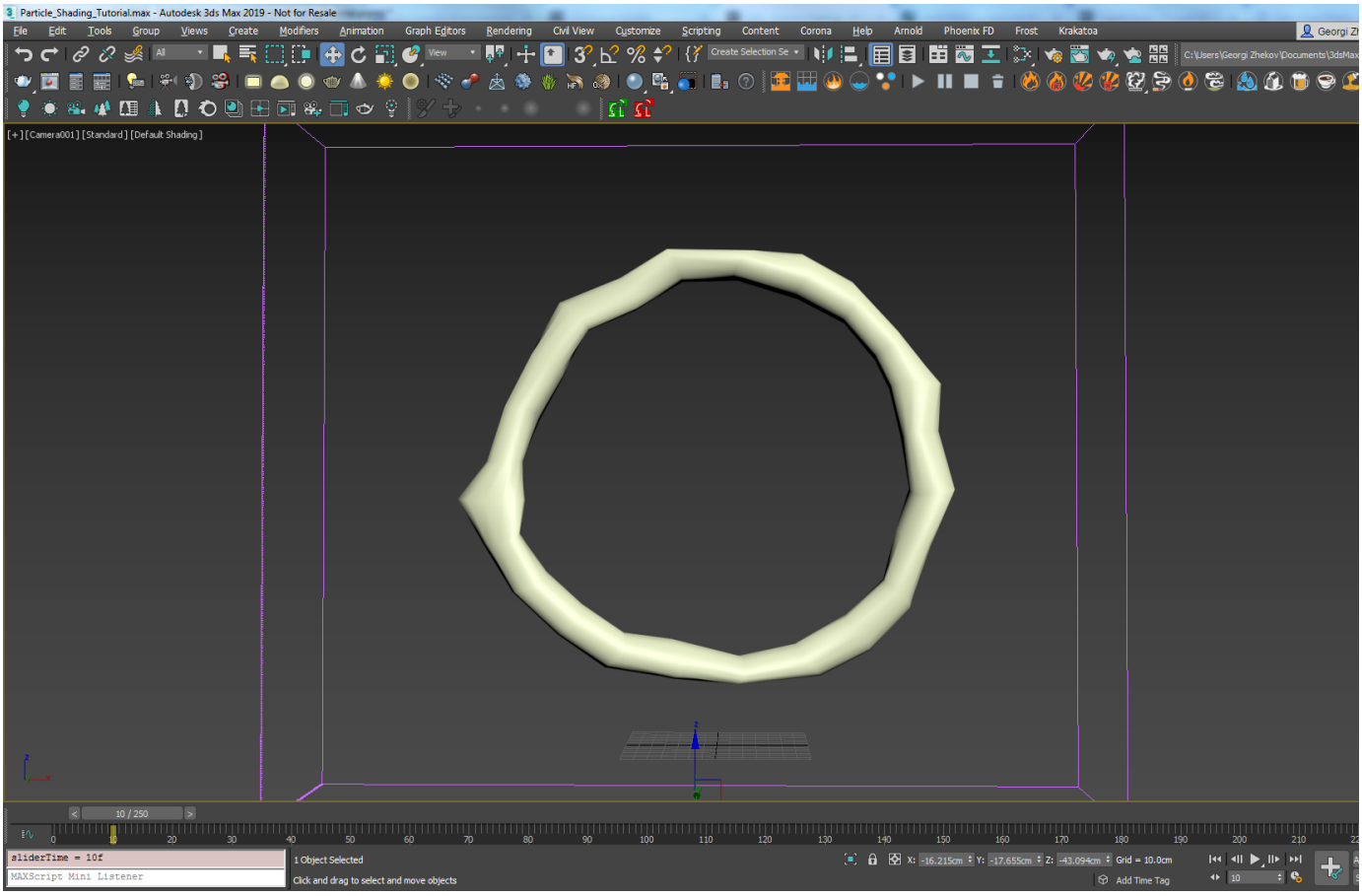


Phoenix Setup

Create a **Phoenix Liquid Simulator** and set the **Grid Cell Size** to **1.25**.

Set the **Size X / Y / Z** to **491 / 142 / 417** respectively.

Set the **Scene Scale** to **10** - this would make our liquid swirls move a bit slower.



Create a **Phoenix Liquid Source** and **add the torus to the Emitter Nodes** list by using the **Add button**.

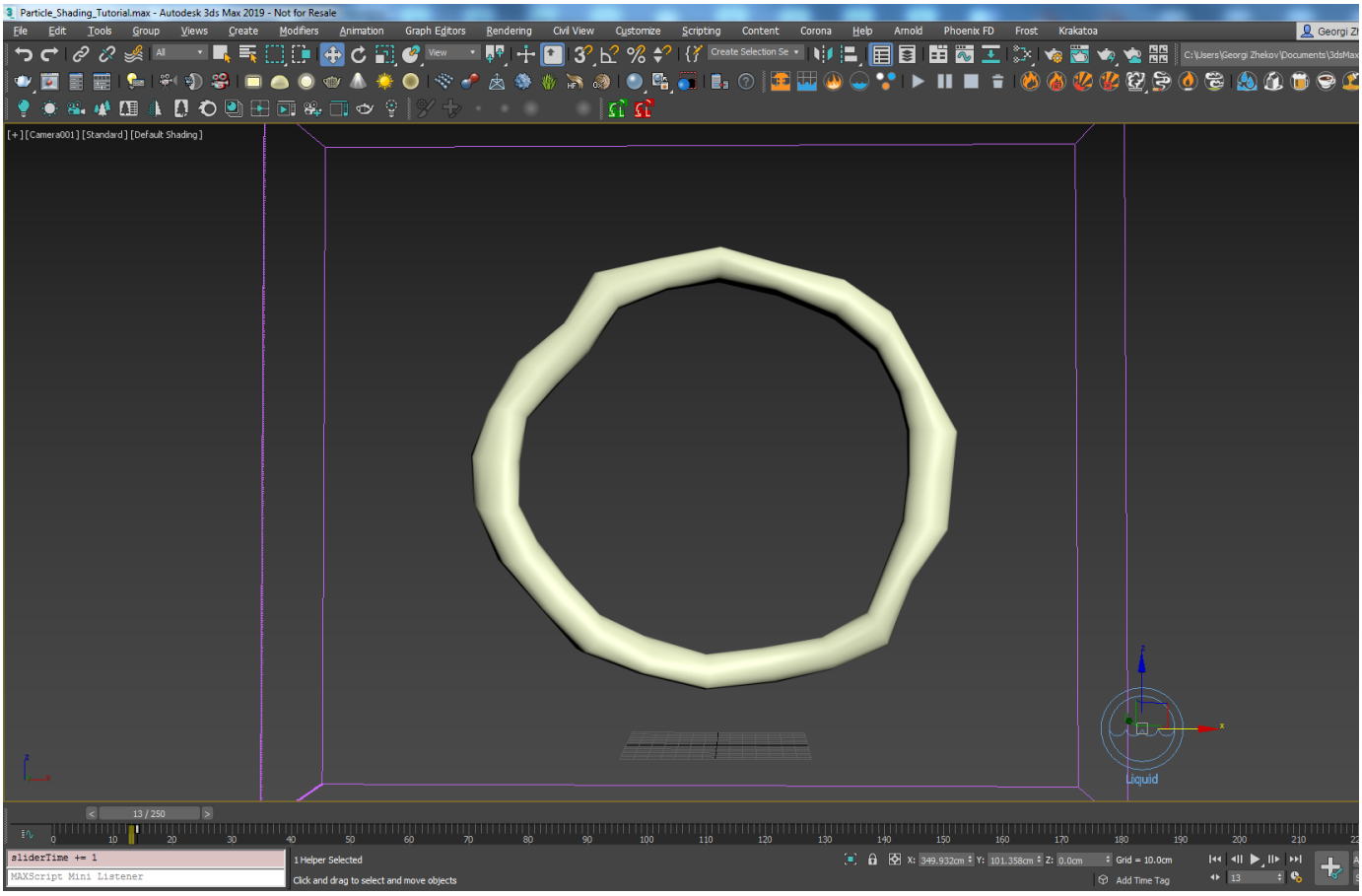
Set the Emit mode to **Volume Brush** (you will be prompted with a message box letting you know that in order for the Volume Brush mode to work you need to set the object to **Non-Solid**, click on the **Make Non-Solid** button) and animate the Brush Effect from **100%** at frame **13** to **0%** at frame **14**. Using the Volume Brush mode the volume of the emitter object will be gradually filled with liquid and in this way giving us a lot more particles to work with.

Turn on the **RGB channel** and plug a 3ds Max **Noise texture** in the map slot.

For the Noise texture set the Source to **Object XYZ** and the Noise type to **Fractal**. Set the **High value** to **0.57** and the Low value to **0.5**. Set the **Levels** to **6** and the **Size** to **120**.

For the **Color #1** of the Noise choose **214, 255, 255** for Hue, Saturation, Value respectively and for **Color #2 161, 255, 255** for Hue, Saturation, Value.

Now that all the required elements for a liquid simulation are present ((1) **Emission Geometry**, (2) **Source** and (3) **Simul ator**), we can run the simulation to see what we've got.



Here's how the simulation looks at the moment. The liquid starts falling down and doesn't stick to the animated Torus geometry.

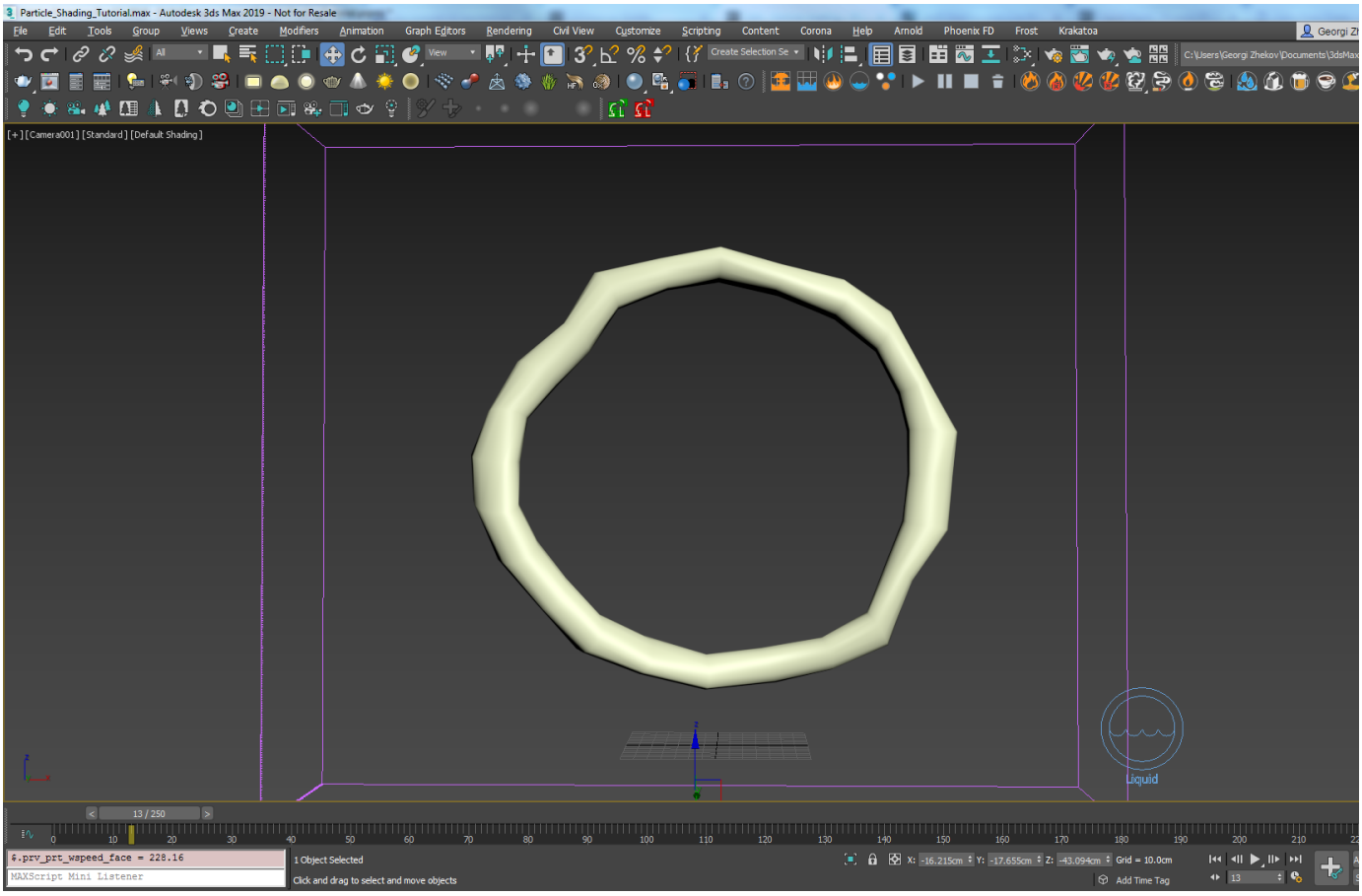
We need to disable the **Gravity** and add a **Body Force** that will pull the liquid towards the Torus.

Phoenix Simulator Setup

Select the **Phoenix Simulator** and from the **Dynamics** rollout disable the **Gravity** checkbox. This way once the liquid is emitted it won't start falling down due to the gravity.

As later we would want to shade our particles based on the liquid channels we need to set those for export to the cache files.

In the **Output** rollout tick the checkboxes for **Velocity** and **RGB** for both the **Output Particles** and the **Output Grid channels**.

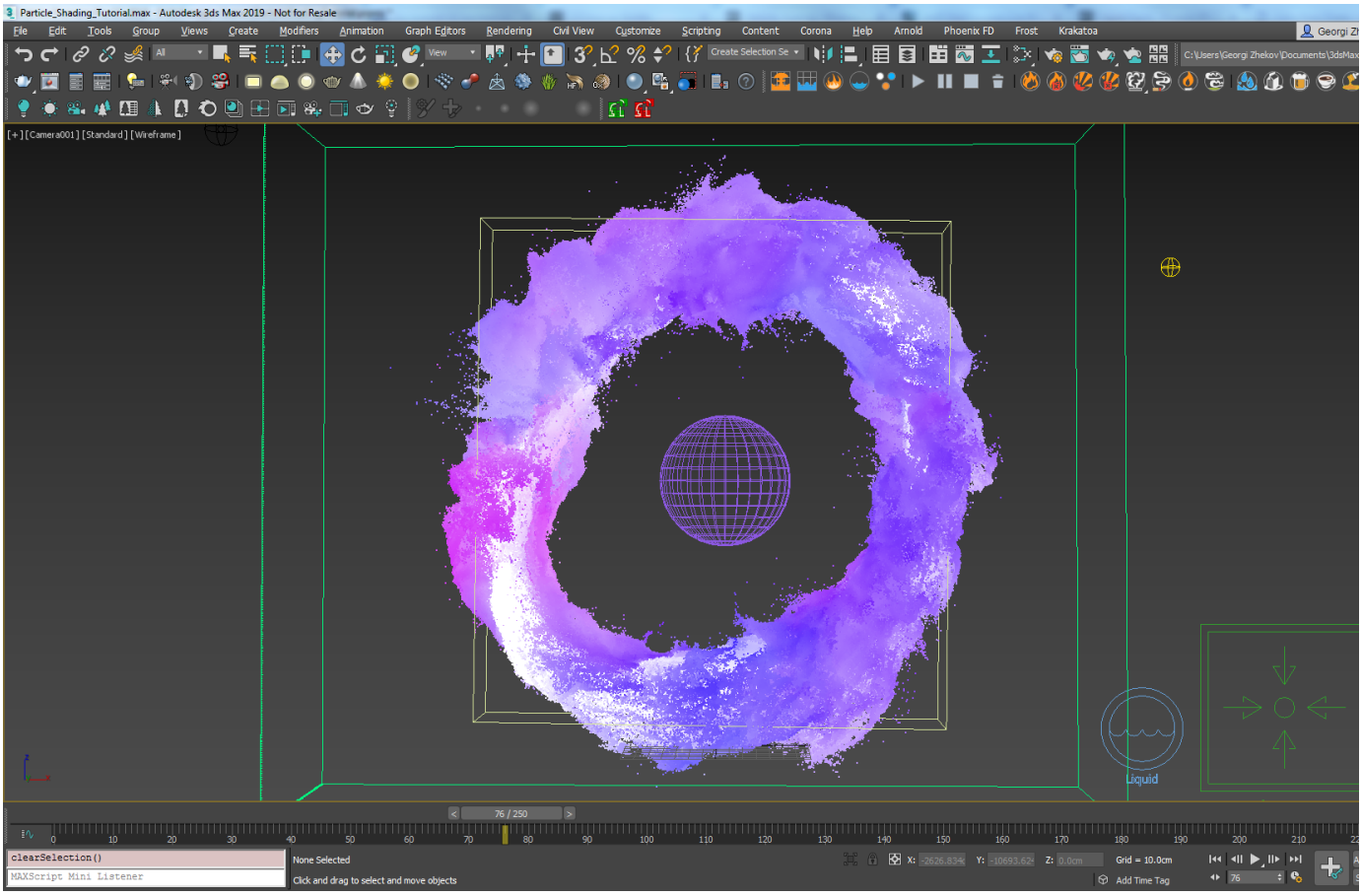


Adding the Body Forces

Now that our liquid is emitted and is no longer falling down we want it to follow the movement of the animated torus.

Create a **Phoenix Body Force** and set the Torus for the **Body**.

Set the **Strength** to **300** and animate its value from **300** at frame **140** to **100** at frame **145**.



Now the particles are rotating and follow the Torus, but that gets a bit boring after a while so we will create another object to attract the particles and make them disappear.

Create a **Standard Primitives Sphere**. Set the **Radius** to **50**.

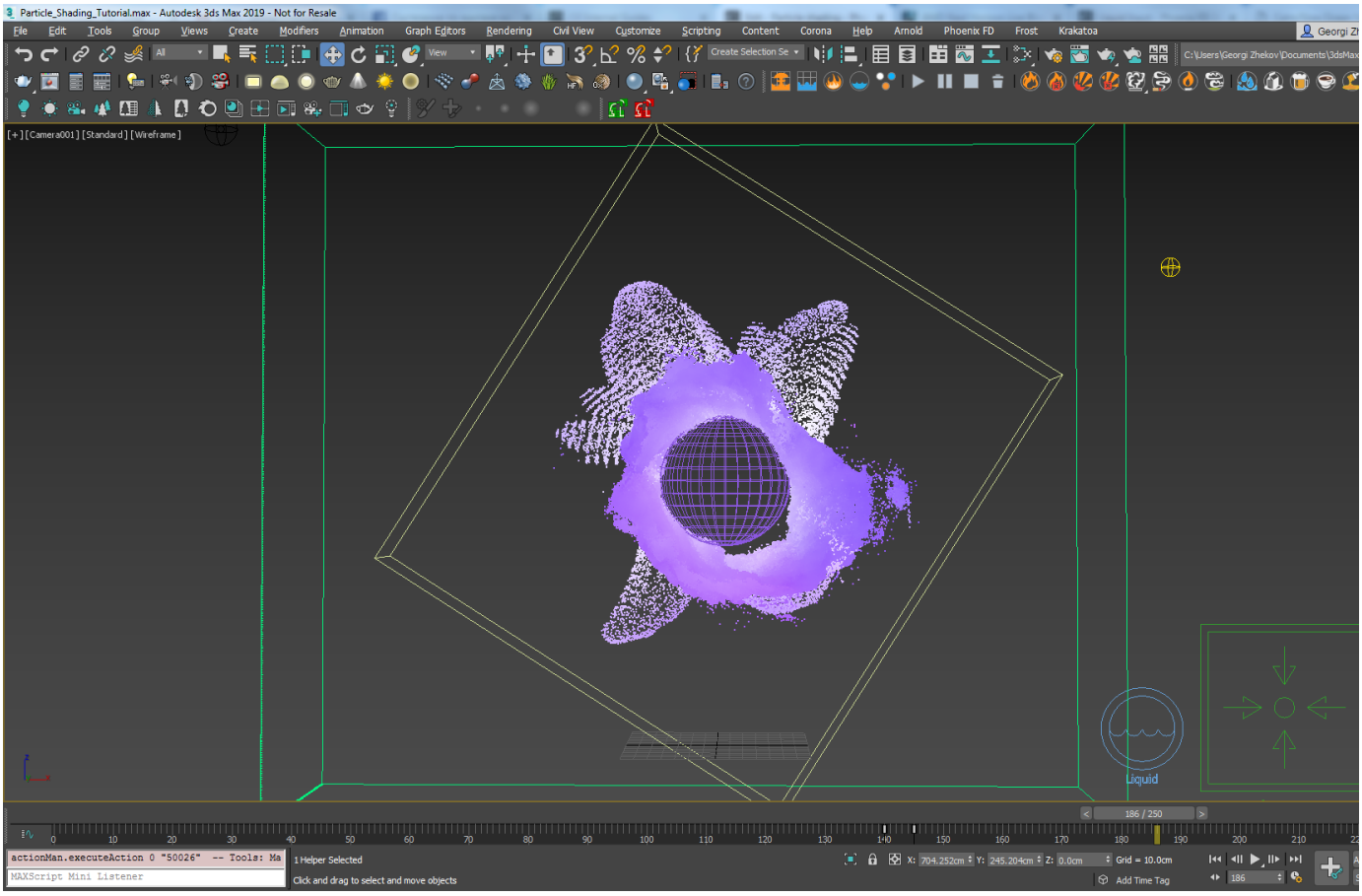
Right-click with the Sphere selected and from the Chaos Phoenix Properties set it to **Non-Solid** so that particles can get inside of it.

Then Turn on the **Clear Inside** option - this way all the particles that get inside of the sphere's volume will be killed.

The only thing left is to add another Body Force that will pull the particles inside of the sphere.

Create a **Phoenix Body Force** and set the Sphere for the **Bo dy**. As we don't want this force affecting the simulation from the start we will need to animate its **Strength** value.

Animate the **Strength** from **0** at frame **140** to **800** at frame **145** .



Here's the result with the both Body Forces affecting the simulation.

Particle Shading

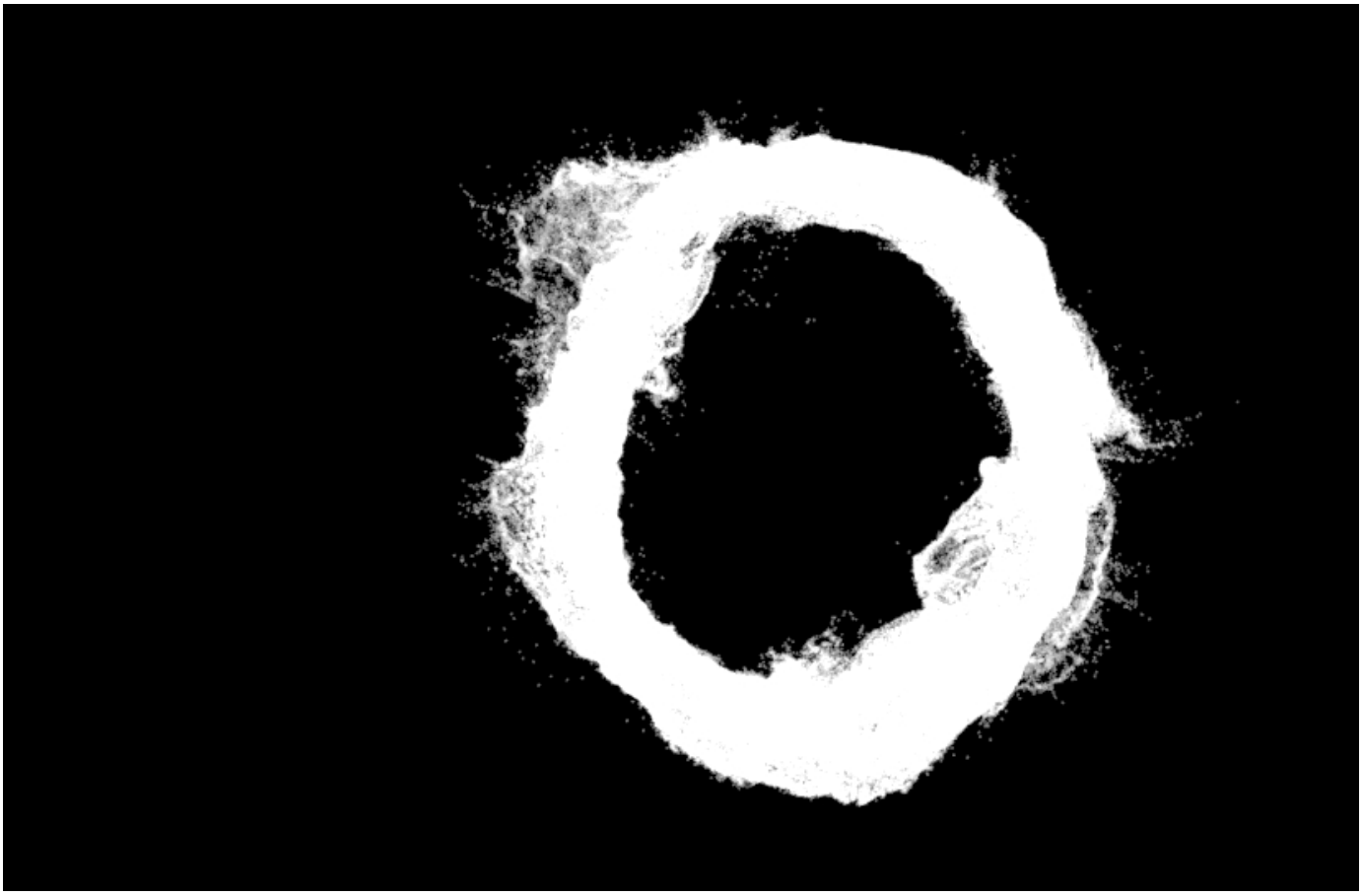
Now that we have the scene setup we need to render it out. By default the Phoenix Liquid Simulator will be rendered as mesh, though in this case we would like to render out the simulation as particles.

Select the Liquid Simulator, right-click and from the Object Properties disable **Renderable**.

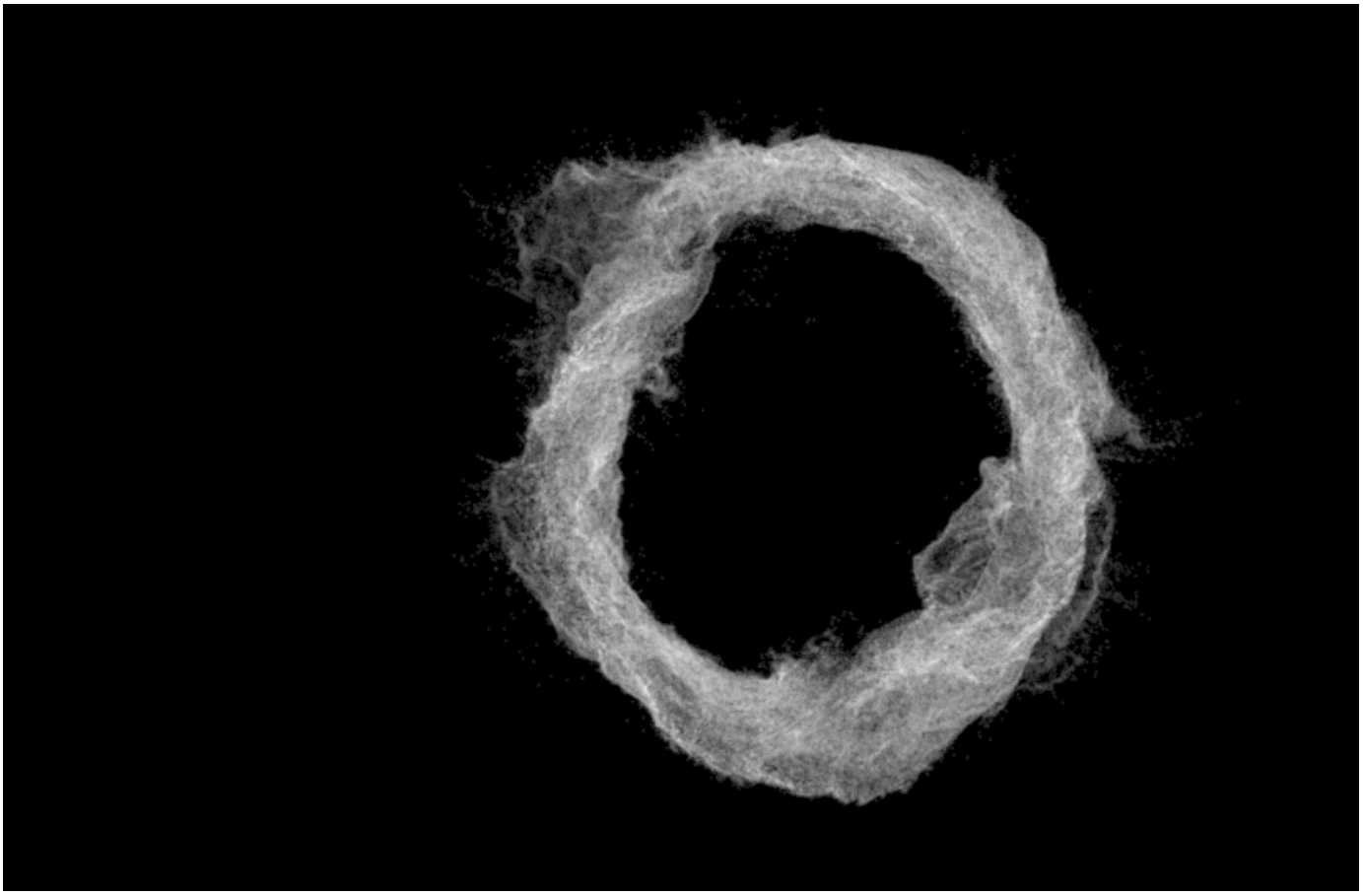
Create a Phoenix **Particle Shader** and from the Add button select the Simulator and pick the Liquid particle group.

If we hit the render button now you will notice that the render is blank. In order to see something we will need some lights.

Create a **V-Ray Ambient Light** and then set the **Color** to **0, 0, 255 Hue, Saturation, Value** and the **Intensity** to **2**.



You will notice that the particles look too dense and burn out. In order to bring back some of the detail let's select the Particle Shader and set the **Point Alpha** to **0.01** and the **Shadow Strength** to **0**.

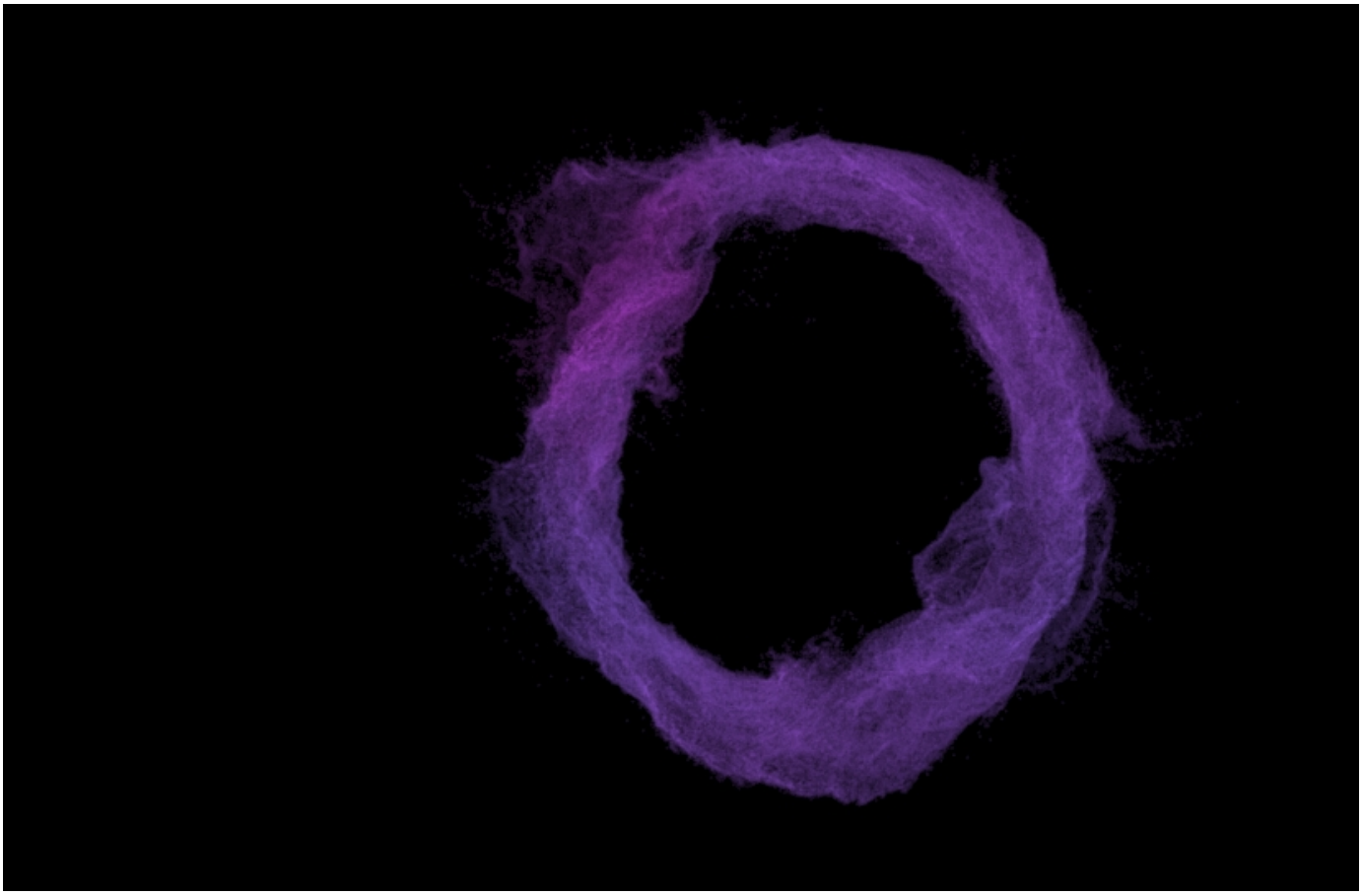


As we have already simulated the particles' **RGB channel** - we would want to use that for the color of our particles. In order to do that we would need a way to read this channel and tell the Particle Shader to use it.

Select the Particle Shader, enable the **Color Map** checkbox and in the map slot plug a **Phoenix FD Particle Texture**.

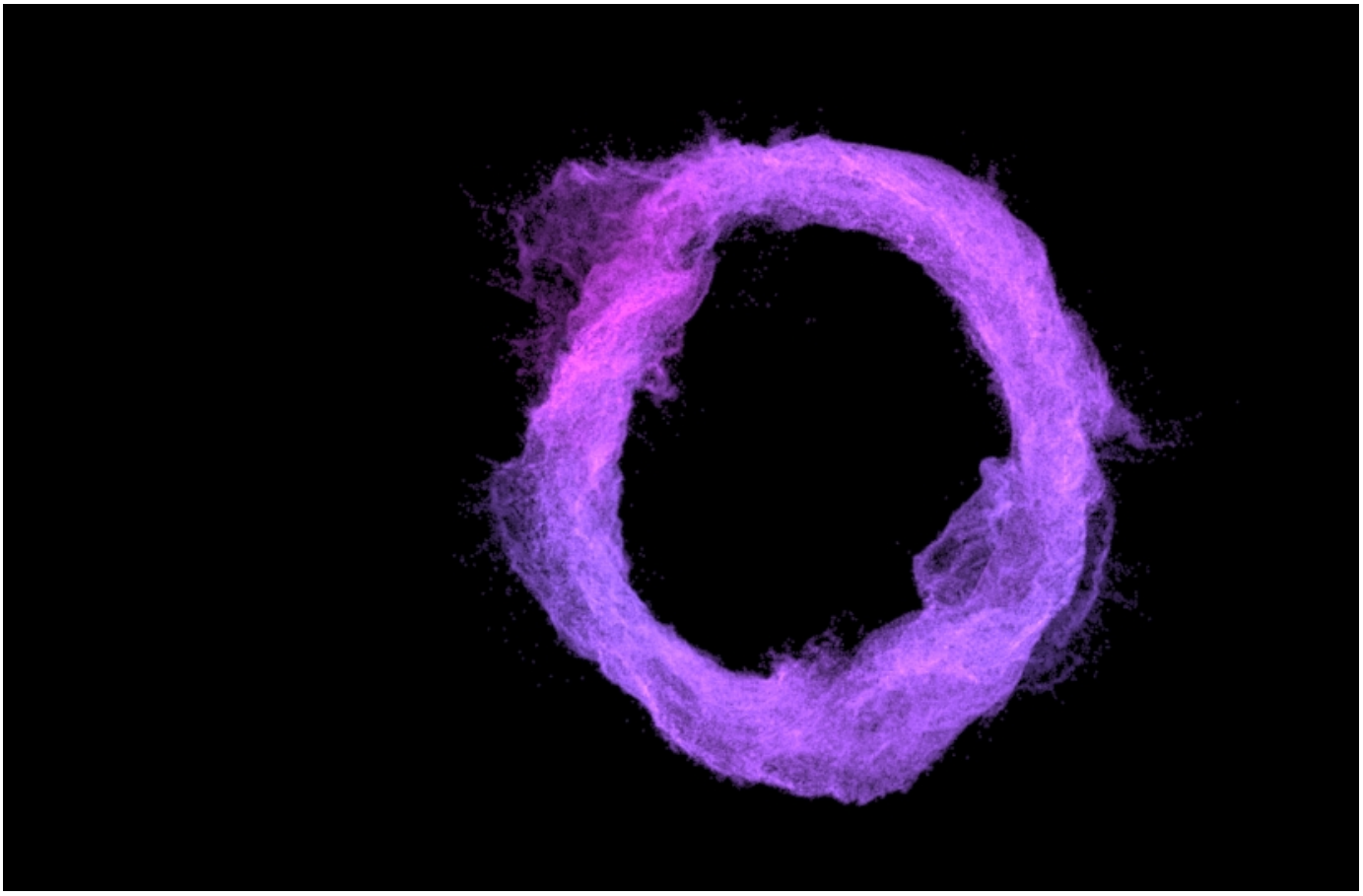
From the Particle Texture options click on the button for the Source Particle System and select the Simulator, then pick the Liquid Particle group.

Turn on **Color From Particle Channel** and select the **RGB** channel. If we render out the result looks like this.



The particles are shaded by the RGB channel, though the color looks a bit pale. We can fix that by increasing the **Color Intensity** inside the Phoenix Particle Texture.

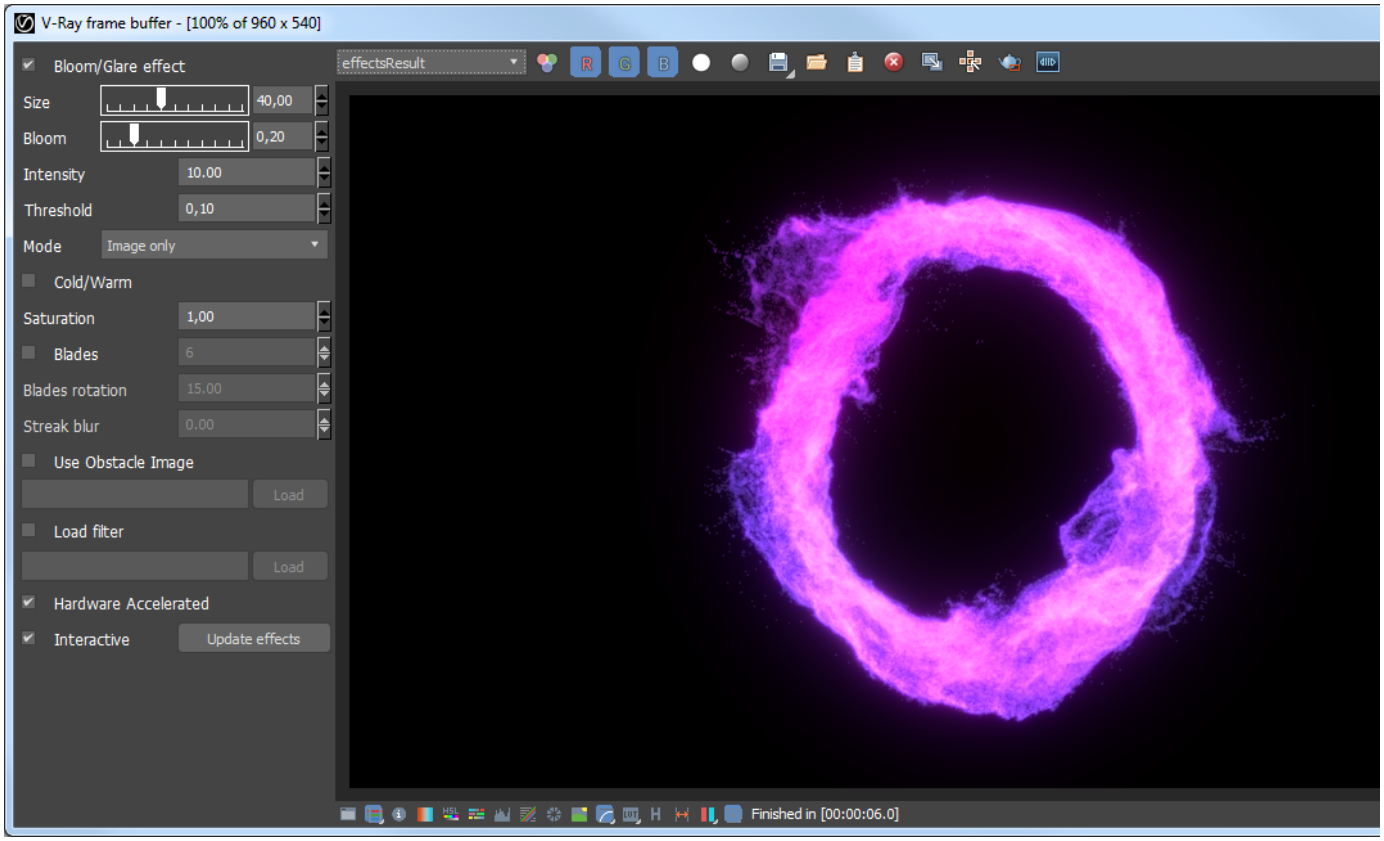
Set the **Color Intensity** to **5** and render again.



Now the particles look much better but still look too ordinary. Let's add some Bloom and Glare from the VFB Settings and spice up those particles.

Open the **Bloom/Glare effect** options and set the **Size** to **40**, **Bloom** to **0.20**, **Intensity** to **10** and the **Threshold** to **0.10**.

We're using the updated V-Ray Frame Buffer coming with V-Ray Next for 3ds Max, update 1. You could achieve the same effect using the Exposure controls on the old Frame Buffer in addition to the Bloom effect, or any compositing package such as After Effects or Nuke.



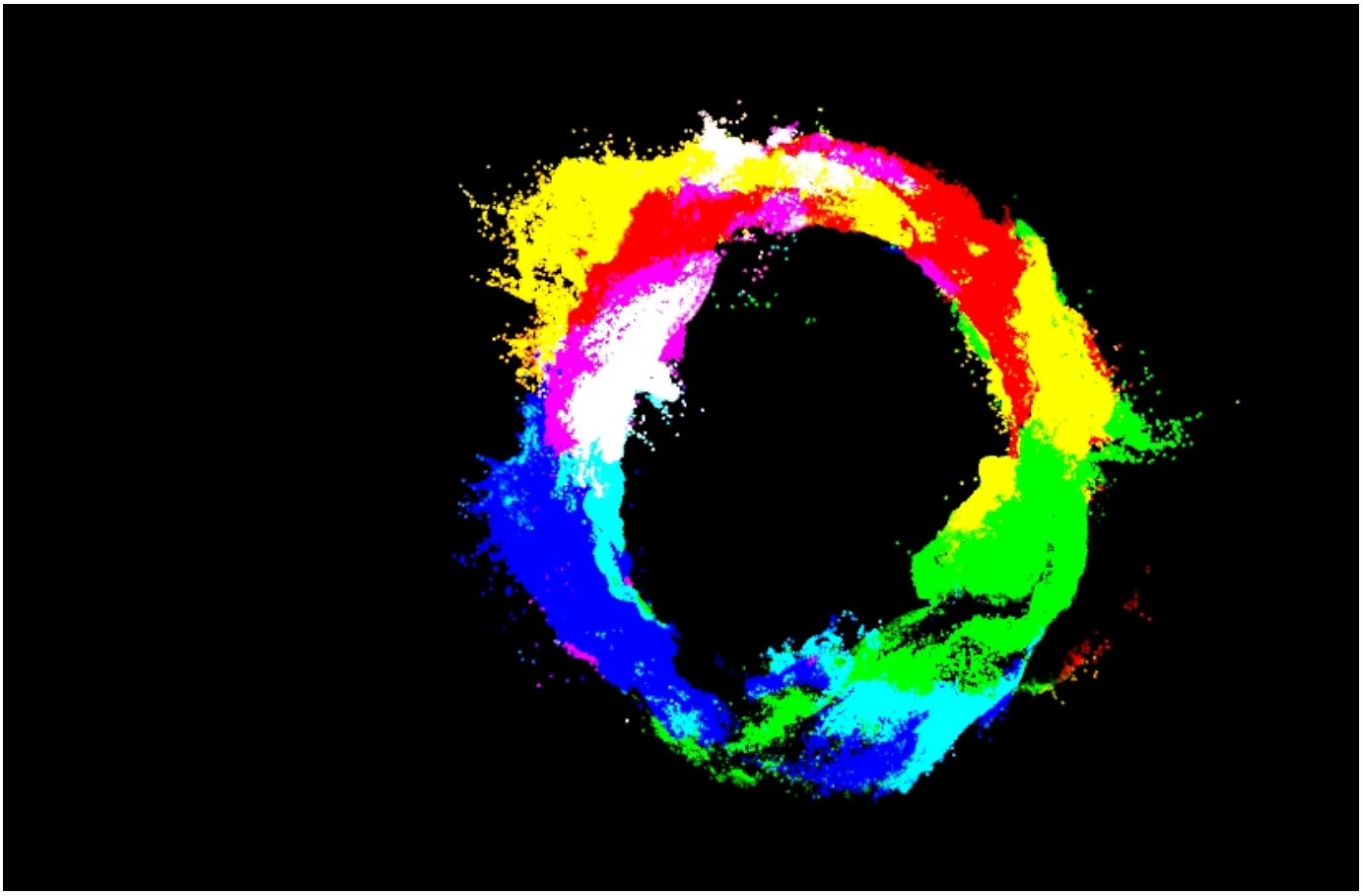
Here's how the result looks in motion.

Particle Shading By Speed

Let's say we want to shade the particle color by speed. How do we do that?

From the Phoenix Particle Texture option set the **Color From Particle Channel** to use the **Velocity** channel.

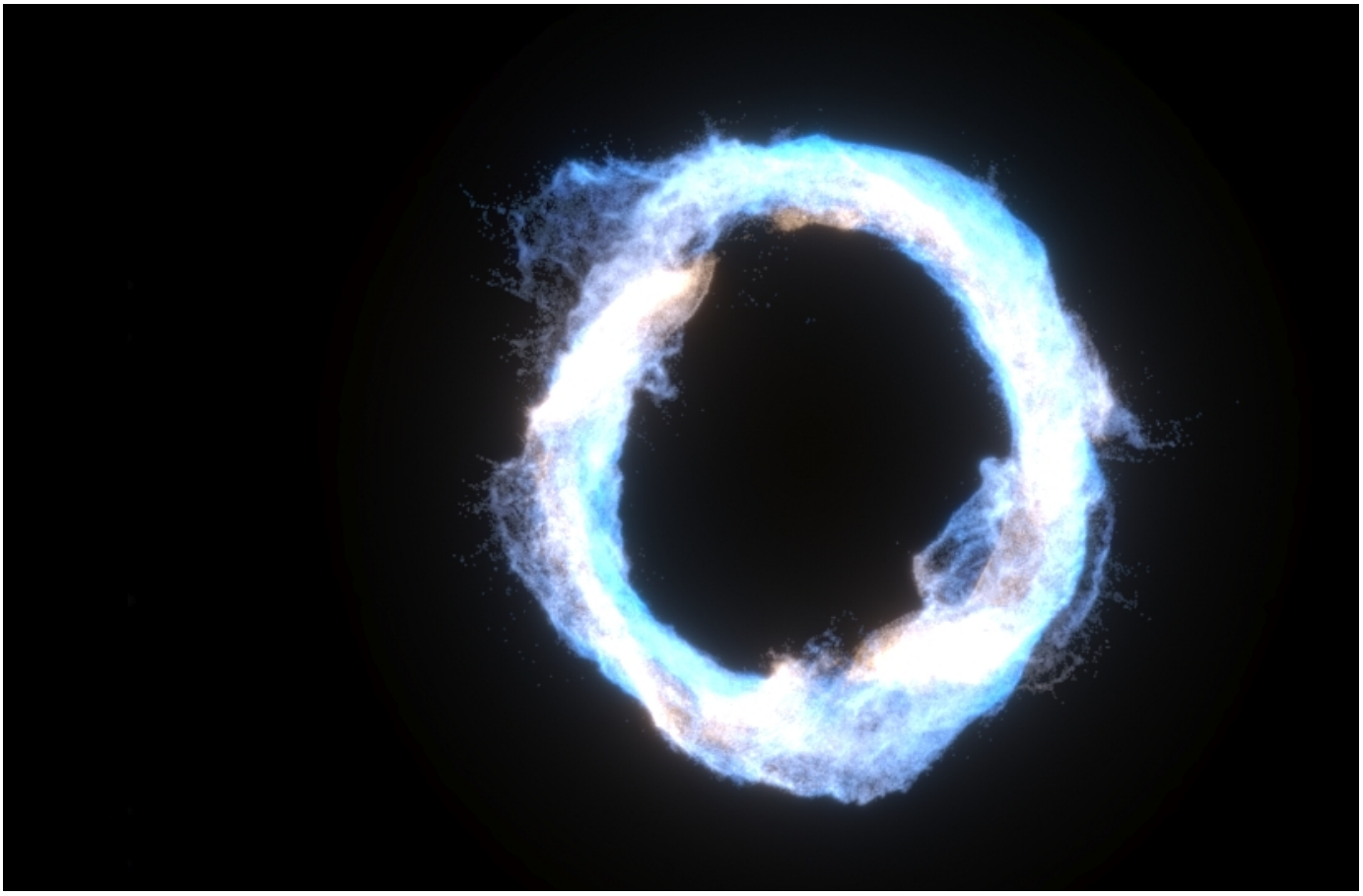
Rendering the result looks like this - not quite what we're after. It currently colors the particles using the velocity vector and the result is similar to what you will see in a 'Normals' render element.



Turn on **Remap Color** from the Particle Texture options and set the **Use Color Component** to **Length**. This way the texture will use the speed of the particles, which is the length of the velocity vector.

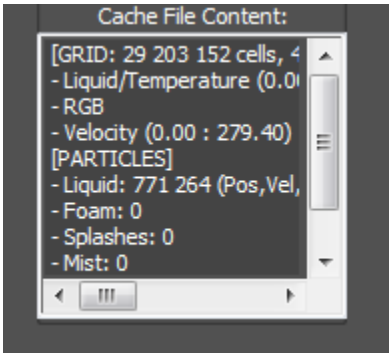
Double click on the first color point to set its color and use **0.6**, **1**, **1** for Hue, Saturation, Value.

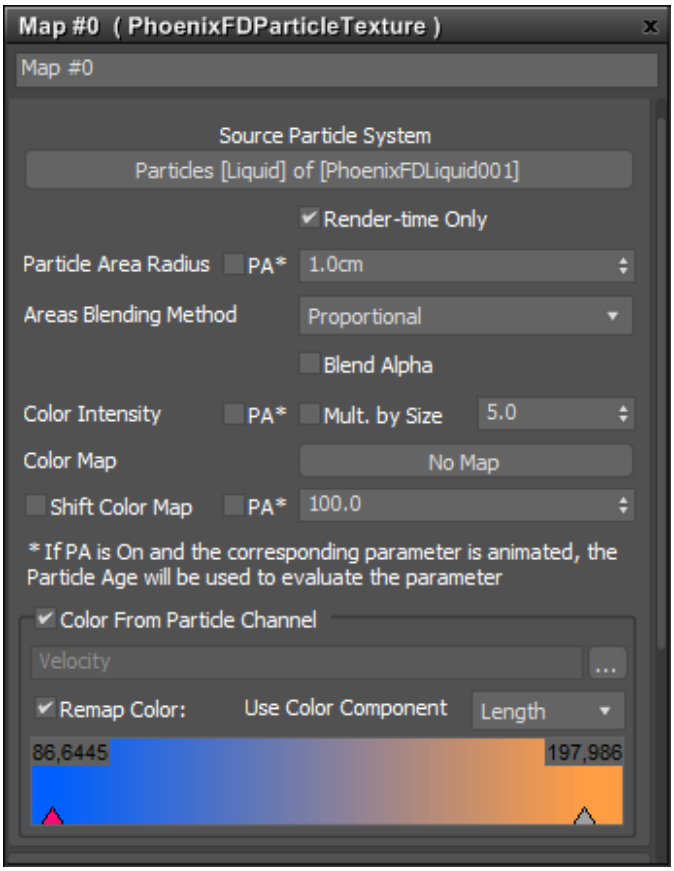
Double click on the second color point and set the color to **0.08** , **0.734**, **1**.



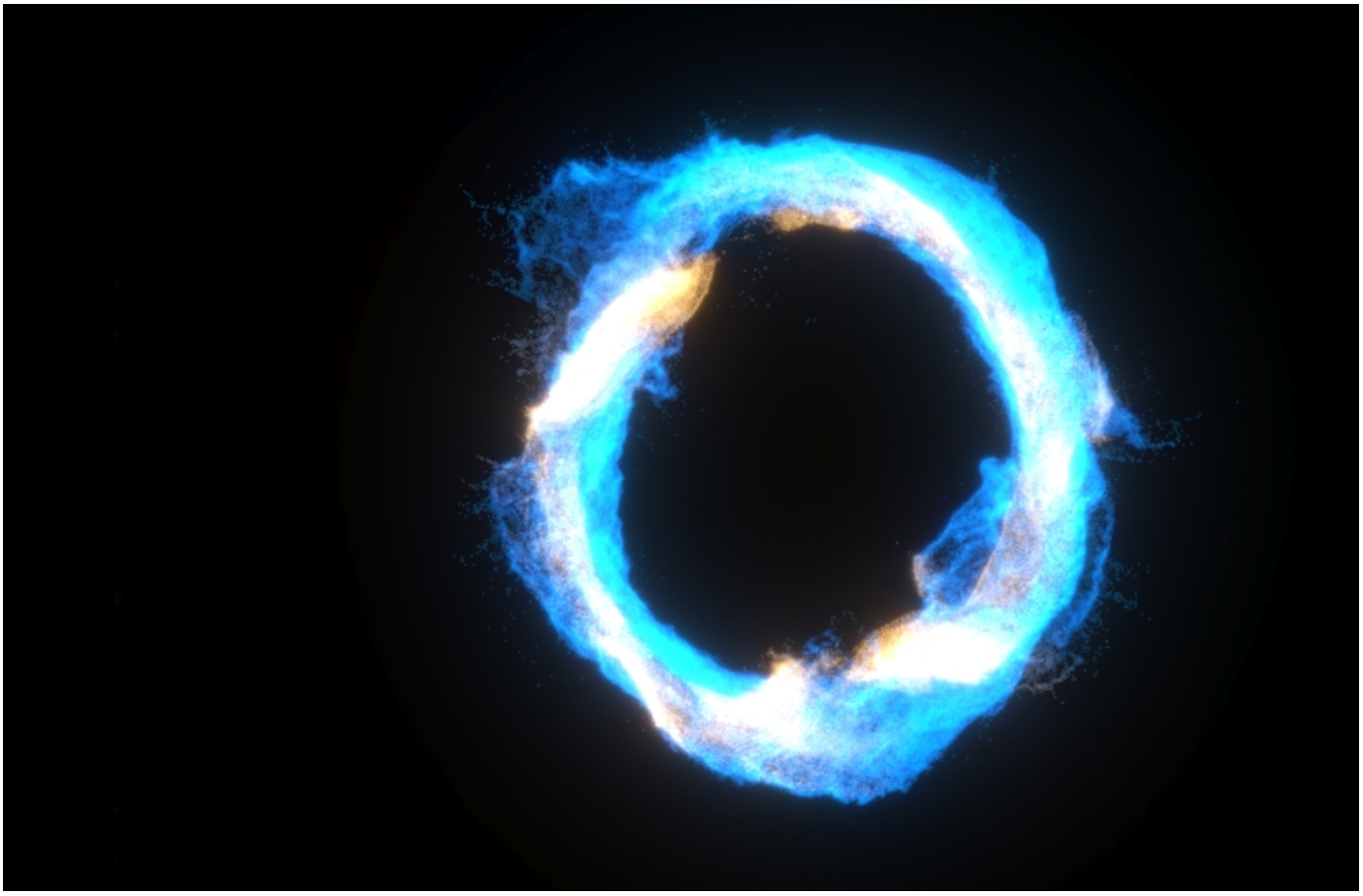
Moving the points, we can control how to colors are remapped based on the speed.

In order to figure out what the real velocity range is you can open the **Simulation** rollout of the Liquid Simulator and inside the **Cache File Content** check the range values for the Velocity channel. In this case it's from **0** to **279.40**.





Move the first color point to position of **86** and the second one to **198**. This way the gradient will tighten up and give the result a bit contrasty look. Note that the numbers displayed on the color gradient are the leftmost and rightmost points on the gradient, and not the positions of the color markers.



Here's how the result looks in motion.

Another really cool thing that you can do with this approach is to combine both methods we have shown above.

So you can take the Particle Texture that reads the RGB channel and multiply it by the Particle Texture that reads the Speed of the Particles.

Duplicate the Particle Texture that we used for the Speed example above, set the **Color from Particle Channel** to **RGB** and turn off the color remapping.

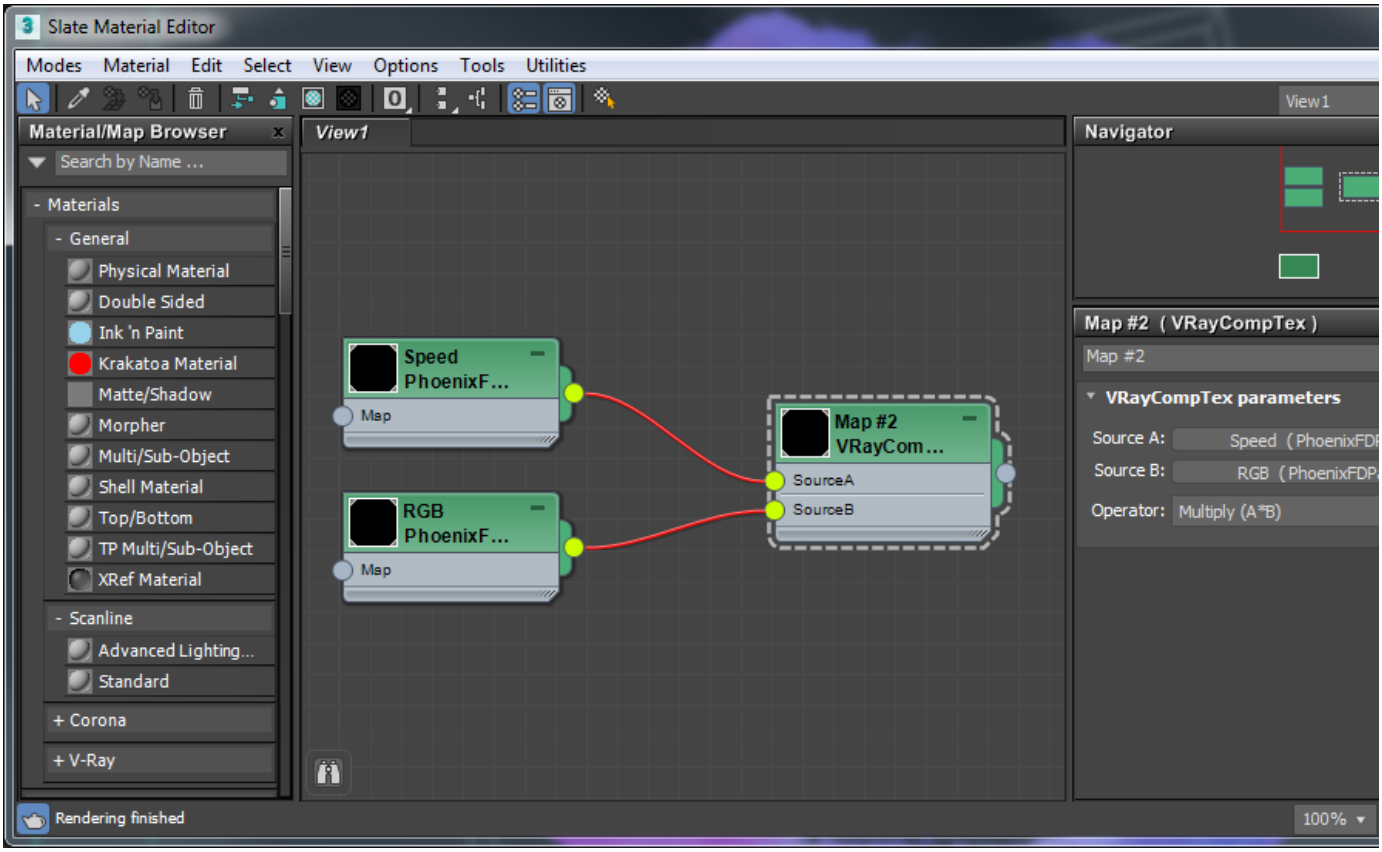
In order to get a bit different result let's change the **Color Remap** gradient for the Speed Texture.

Set the first point to **0**, **0**, **0.1** Hue, Saturation, Value and move its position to **70**. This sets the color to a dark gray and when we later multiply this color by the RGB texture some of the color will get through, but with a lower value.

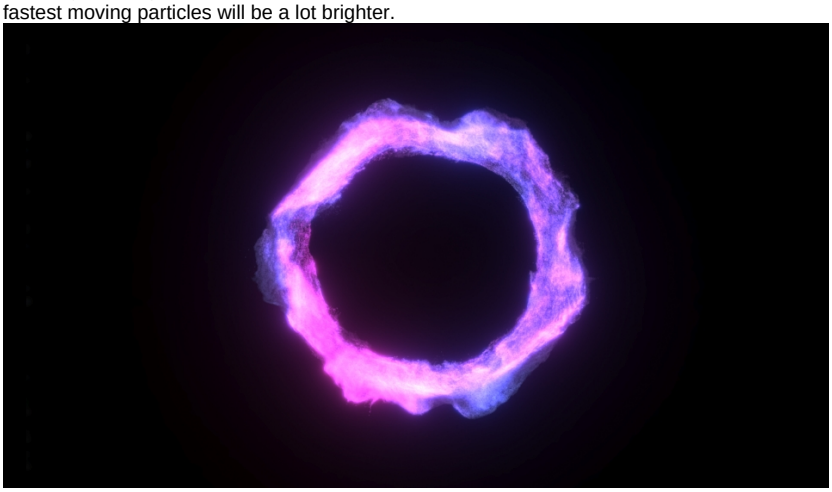
Set the second point to **0**, **0**, **1** Hue, Saturation, Value and move its position to **100**.

Create a **V-Ray Comp Texture** and connect the **Speed** and **RGB** particle textures to it.

Then set the Operator to **Multiply** and set the **V-Ray Comp Texture** as a Color map for the Particle Shader.



This way we do get the colors from the RGB channel but the fastest moving particles will be a lot brighter.



Let's modify the overall look and make the particles a bit denser.

Select the Particle Shader and set the **Point Alpha** to **0.11** - this will bring the particles' opacity back up.

Set the **Shadow Strength** to **5** - this way we'll get stronger and darker shadows.

Now let's add some more V-Ray Lights.

Select the V-Ray Ambient Light we have already created and disable it.

Create a V-Ray Sphere Light and position it at **X: 300, Y: -266, Z: 530**. Set its **Multiplier** to **900**.

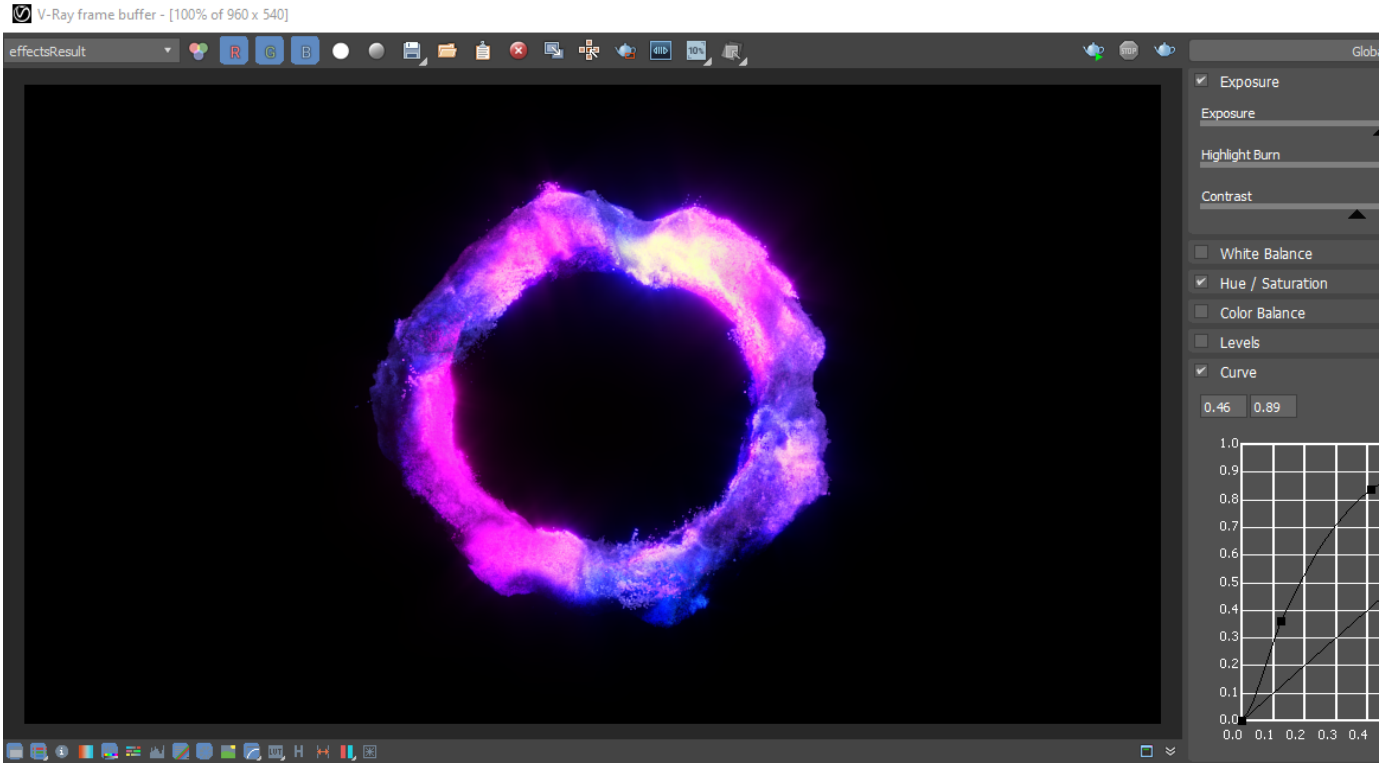
Create another V-Ray Sphere Light and position it at **X: -460, Y: 230, Z: 530**. Set its **Multiplier** to **200**.

And this is how the final result looks like.

In addition, you can enable the Exposure and Curve options from the VFB Settings to make the render even more interesting.

Open the **Exposure** options and set the Exposure to **0.2**.

Experiment with the **Curve** to achieve the desired look.



And here is the final rendered result.