AUTODESK UNIVERSITY 2013

Particle Flow, the Motion Graphics and VFX Toolkit

Anselm von Seherr-Thoss – Incendii LLC VisualEffects

DG3531-L Class Description (use edited version off the AU Website) [Arial 10]

Learning Objectives

At the end of this class, you will be able to:

- Break compound structures using mParticles
- Utilize new operators to create 8-bit style animations
- Use helpers and sound to drive your particle simulation
- Understand the basic concept of both new PFlow additions
-

About the Speaker

I am an award winning Visual Effects Technical Director and VES(Visual Effects Society) member with emphasis on particle based effects as well as Fluid Simulations, Shading and Post Production. I worked on movies like James Cameron's "Avatar", Star Trek 2 - Into Darkness, G.I. Joe-Rise of Cobra, The A-Team, 21JumpStreet, Beautiful Creatures, Ridley Scott's "Robin Hood", Priest, SuckerPunch (PreViz), Beautiful Creatures (R&D), Dragonball: Evolution, Tree of Life (R&D), Clash of the Titans(3D conversion), Niko & The Way to the Stars, "Loewenzahn", Angel Camouflaged, The Big Year as well as numerous commercials and music videos. I consulted vfx on movies like Gulliver's Travel, Skyline, Moving Day (Shortfilm), "Il sogno del Maratoneta(Day of the Race)", "Battle of Vienna".

mail@incendii.com

8-Bit style/motion graphics with ADM (Advanced Data Manipulation)

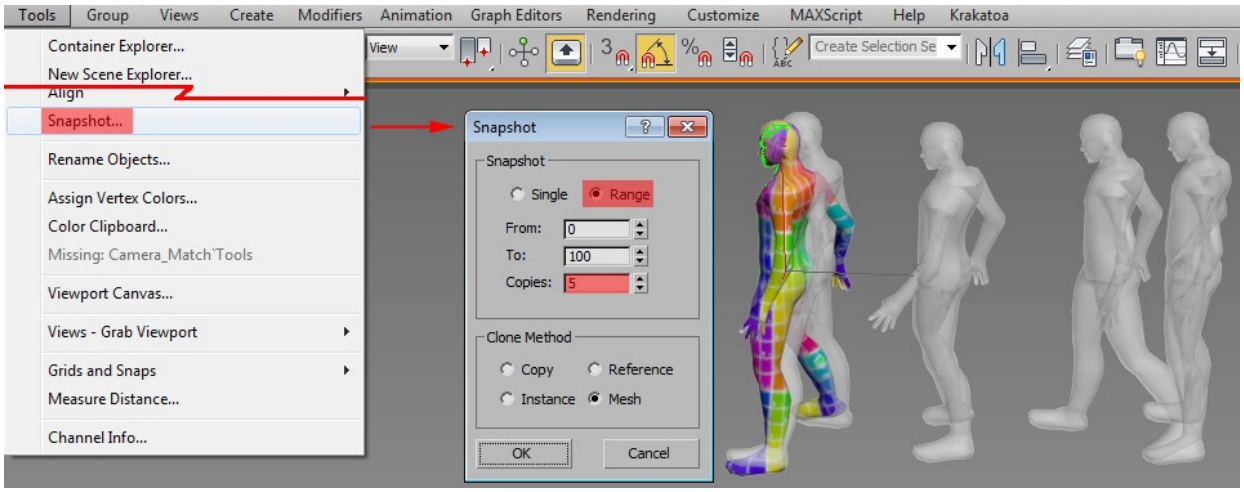
Utilize new operators to create 8-bit style animations

ADM adds a great range of possibilities to PFlow being virtually a node based code toolbox.

The secret for rather speedy workaround is to rule out all particles not needed at any point in time. The 8-Bit style is defined by closed (water tight) objects and their volumes. So it makes sense to check if a particle is inside or outside of a volume first and don't even deal with the outside of volume ones further. That way shapes and shaders only have to be calculated on a fraction of particles.

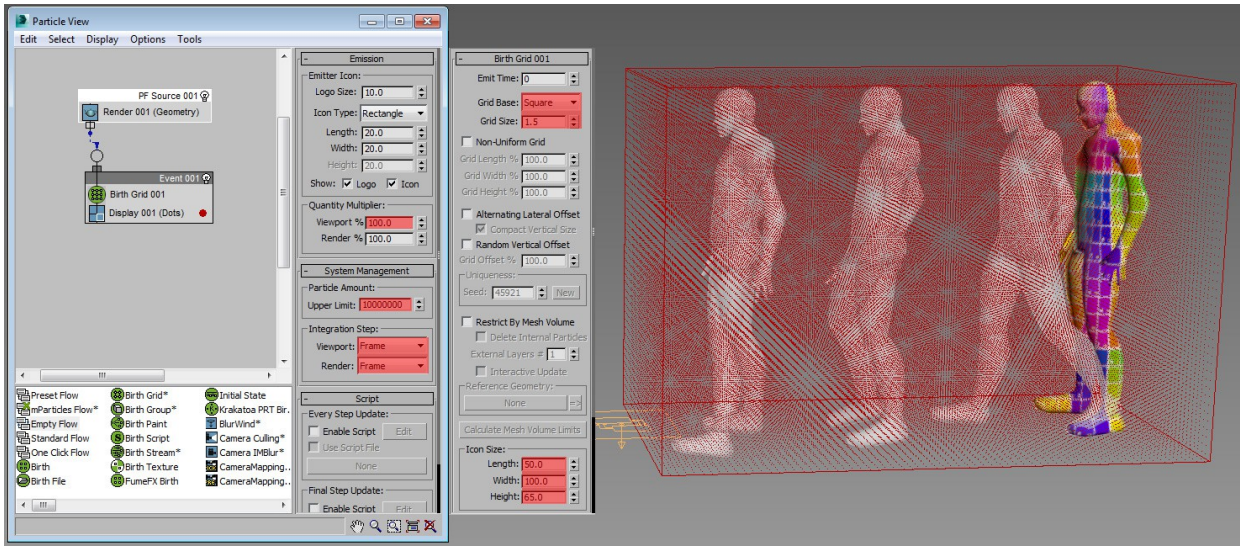
The basic check if a particle is inside an object is a boolean operation which tends to be quick since it can only have two possible outcomes: true or false.

But before we even can do that we need to get a rough idea of the motion path. We can just make snapshots every few frames depending of the motion complexity. ***Tools > Snapshot***:

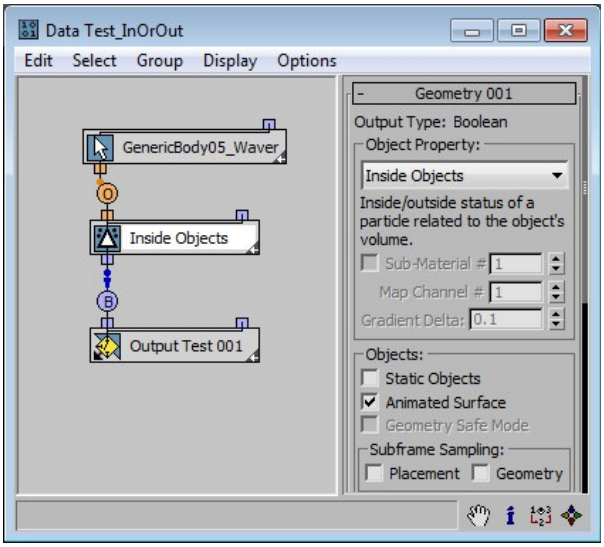


After you have created your snapshots you can create an ***Empty Flow***. Make sure you display 100% of the particles in the viewport and that your ***Integration Steps*** are both ***Frame***. For an 8-bit effect we don't need more precision and it would be a waist of resources. Also make sure your ***Upper Limit*** is well above 100k in case you need the amount of particles!

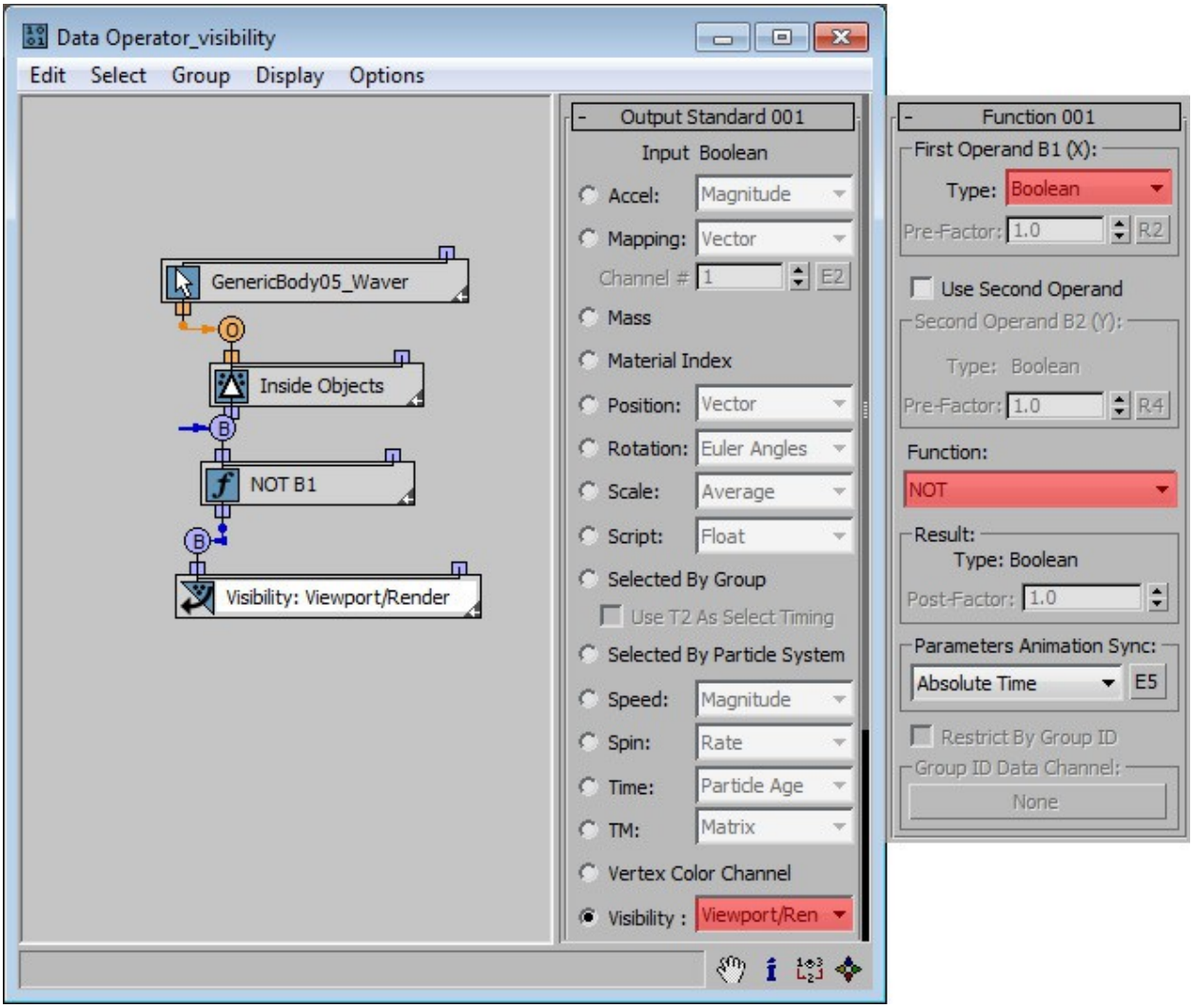
Then create a birth grid that engulfs the motion of your object. Make sure your ***Grid Base*** is ***Square***. Otherwise you will have brick wall type off sets. For still objects we could define a volume in the Birth Grid already! In that case we could check ***Restrict By Mesh Volume*** and pick a mesh but we want to work with motion so this is unfortunately not an option here.



Now that we have a grid filled with particles it's time to sort out what we don't need. Drag a ***Data Test*** under the birth grid from the depot. This is the first ADM part we are going to use. Birth Grid is technically a part of mParticles ;) So we want to extract particles that at inside of a moving, closed scene object. Thus we need ***Select Object*** sub-operator first. In there we can pick one or multiple scene objects to read data from. Pick your object. Next we want to read data. Our object is a geometry object being either an editable mesh or poly (edit meshes are faster to evaluate btw., they carry less data as the newer editable polys) so we need a ***Geometry*** sub-operator. From the drop down menu of properties we pick ***Inside Objects***. Make sure to check ***Animated Surface*** since our object is animated. Lastly we need an output to pipe the results of this Data Test back into the flow. Drag a ***Output Test*** sub-operator into the Data View and connect it. The default ***True To Satisfy Test*** is the default and our preferred value here. Like any other test in PFlow the Data Test will pipe particles into a new event.



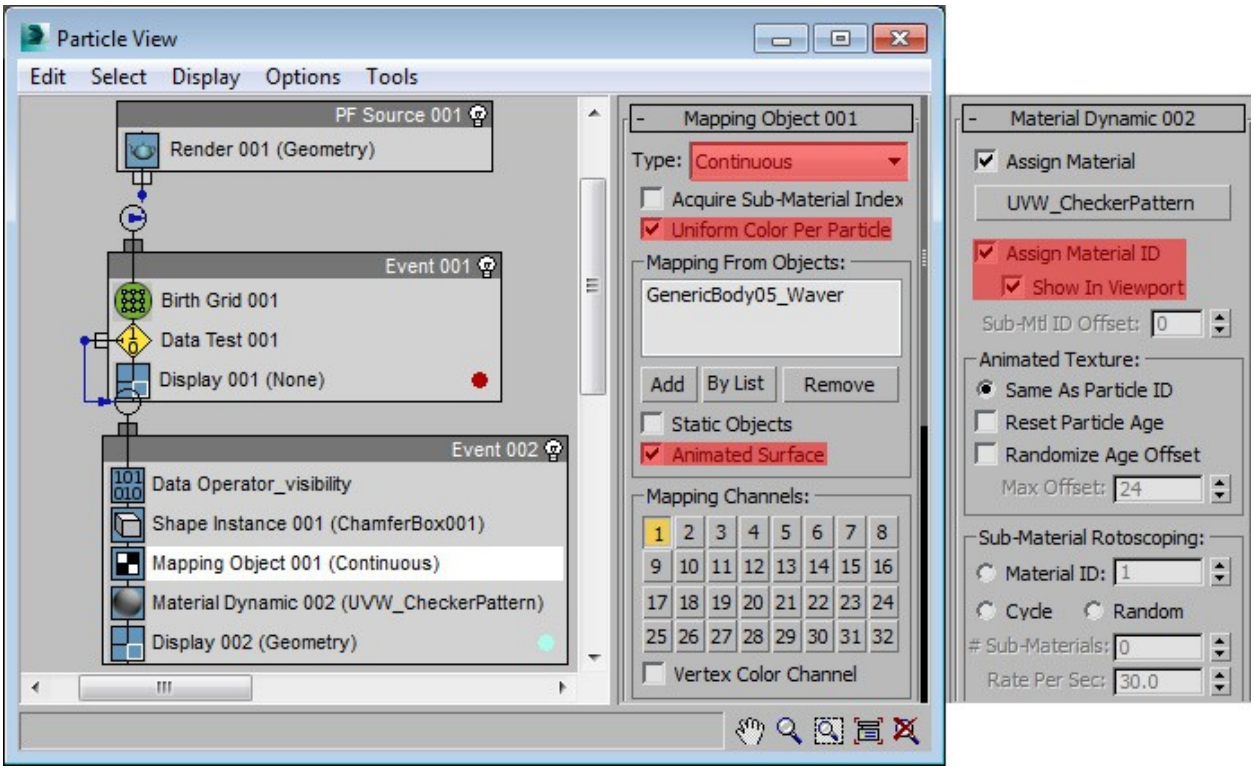
Now comes the part where we make particles outside of the mesh invisible and not renderable. This has to happen in the new event. Create a ***Data Operator*** and drag a ***Select Object*** and ***Geometry*** sub-operator into the data view. We will need to set the geometry sub-operator to ***Inside Object*** again. A data operator has different output types then a data test. We need an ***Output Standard***. From it's pick-able output data sets we choose ***Visibility Viewport/Render*** all the way at the bottom. When you add a ***Shape*** or ***Shape Instance*** operator (adjust size to be slightly smaller than the grid spacing) now you will notice that the full grid is filled with shapes minus the object we are testing visibility with so we need to invert the test result. That's easily done with a ***Function*** sub-operator. Un-check the second operand option since we don't want to do equations of values and set the type to ***Boolean***. As function pick ***NOT***. This means what ever the operator above calculated will be inverted.



This is how your Data Operator should look like by now.

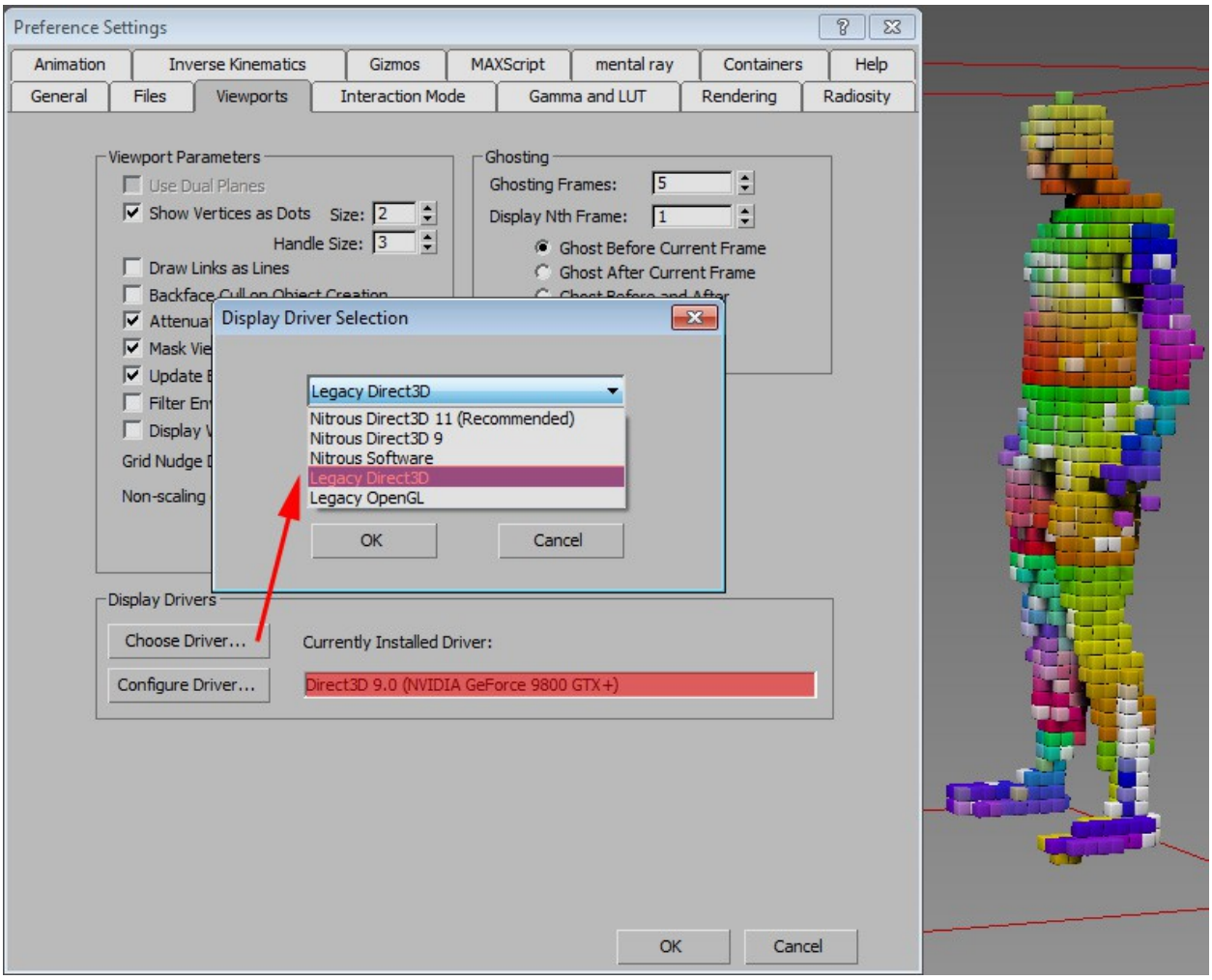
In the next step we are going to inherit the objects UVWs and colors. The oldschool and slower way of doing this would be using a ***Mapping Object*** and ***Material Dynamic*** Operator. Make sure the Mapping type is ***Continuous*** because our object is in motion and UVWs might stretch or compress. Also make sure to only inherit a ***Uniform Color Per Particle***, otherwise you will inherit accurate textures but we want a single color per particle.

Once the UVW mapping is inherited you should add the same shader to the particles as the object has. Just make sure to use a ***Material Dynamic*** since the color of each particle will constantly change. Dynamic material operators are for changing shaders.

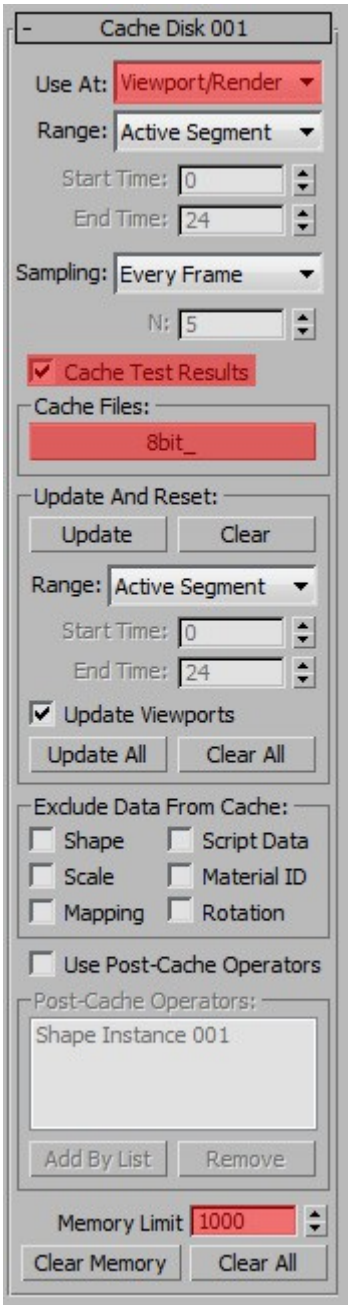


On an important note the NITROUS viewport will not support the Mapped material or vertex color display in either of the two ways shows in this class. So you might want to change the viewport to ***Legacy DirectX9*** in the ***Preferences*** for this tutorial!

Here is how:

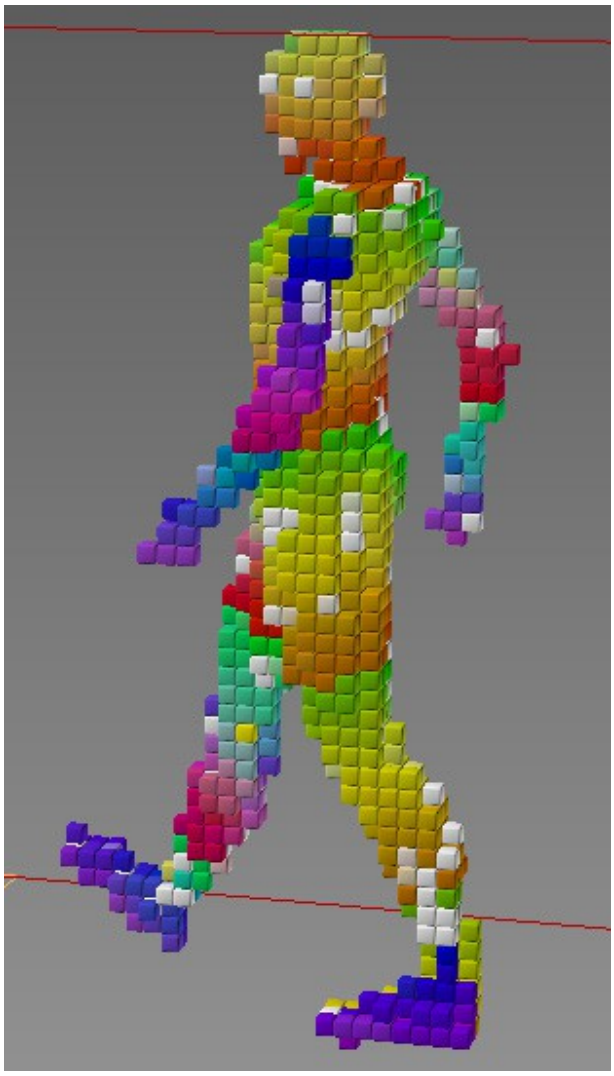


Now that our particles show the inherited mapping and shader in the viewport it's time to cache for the first time! We are going to utilize the ADM ***Cache Disk*** operator which caches every frame onto the HDD. Drag a Cache Disk from the depot into the root event right under the Render operator. Make sure to cache ***Viewport/Render*** settings. Check ***Cache Test Results*** and define a path to cache to. All the way at the bottom of the Cache Disk the ***Memory Limit*** should probably more then 100MB, make that 1000 at least! Once everything is set hit ***Update All*** for the cache to run through.



After the caching process make sure to uncheck ***Cache Test Results***! Otherwise the system will update the cache constantly as you scrub through the timeline. Now your cache should play with pretty decent performance in viewport and rendering.

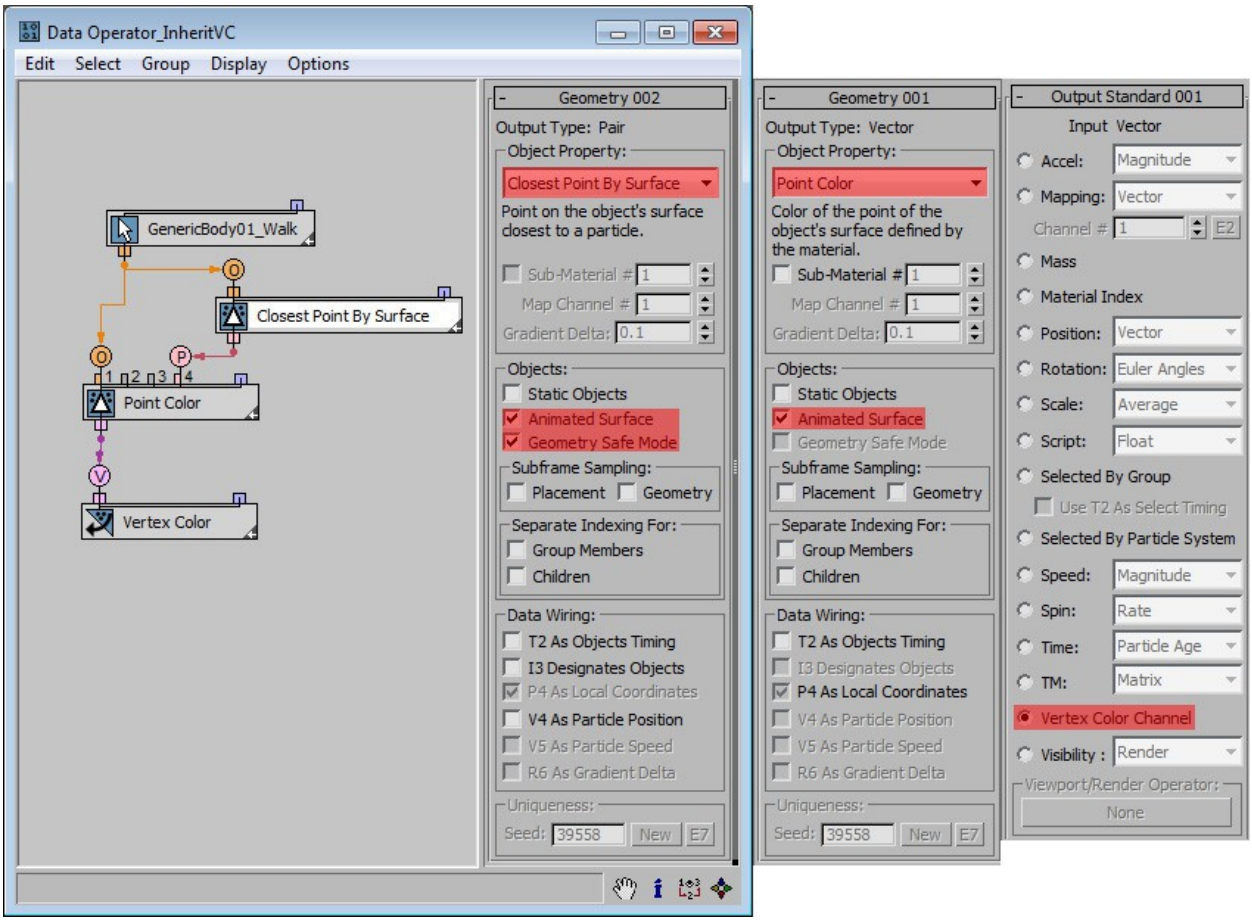
Now that this is done ou should see something like this:



Now it's time to look into a new and faster way using ADM to get similar results!

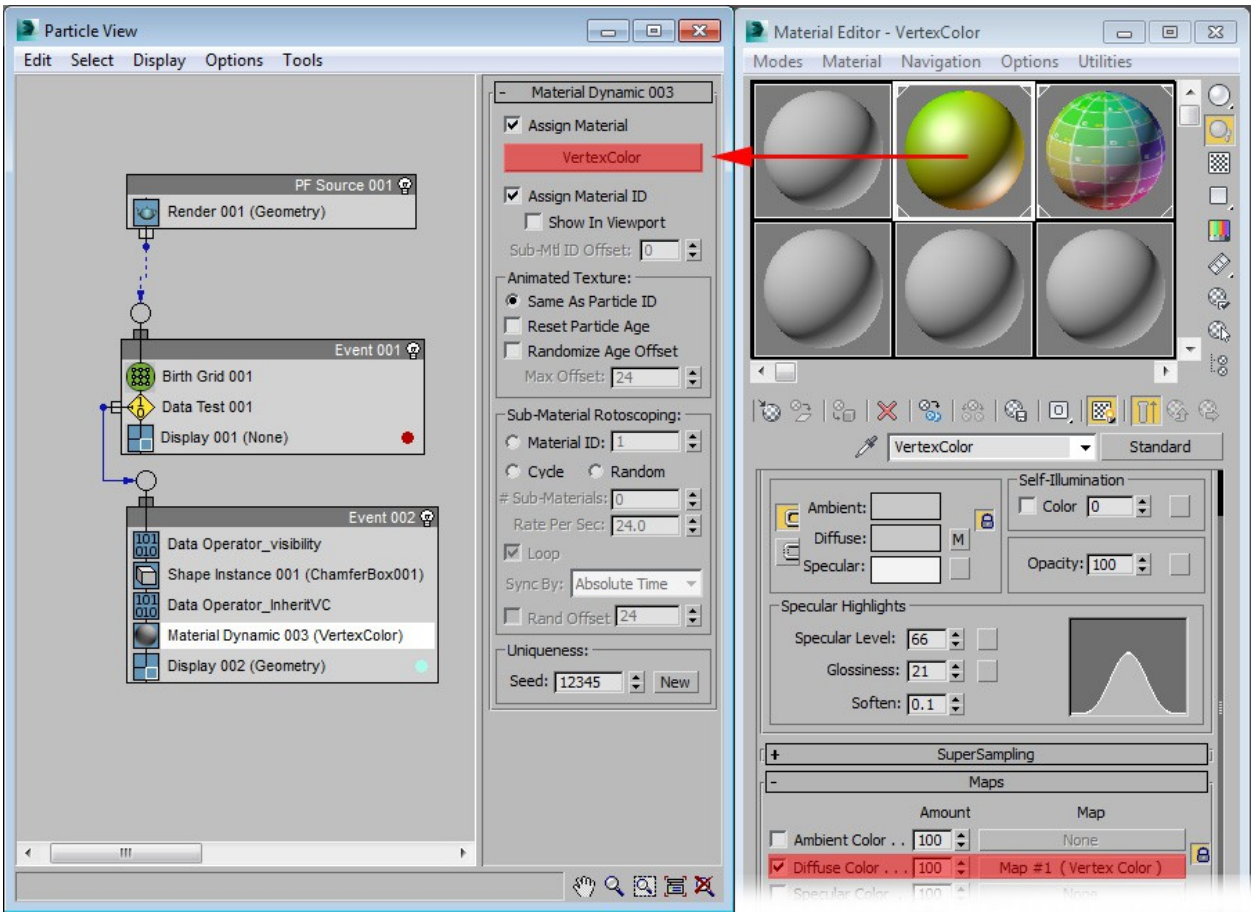
Put the Mapping Objects and Material Dynamic aside for now and create a ***Data Operator*** where the Mapping Object once was in the flow. We start with a ***Select Object*** sub-operator again as we want to read data from a scene object. And just like in the Visibility/Renderable Data Operator we need ***Geometry*** sub-operators again, we are still reading data from a piece of geo ;) However this time we want to read the ***Point Color*** so pick that from the drop down menu. Make sure you check ***Animated Surface*** since our object is constantly moving, this will allow for precise readings. Now Point Color demands two inputs: *WHICH* object to read color from and from *WHERE* on the object. So we need another value here. To define where on the object it needs another ***Geometry*** sub-operator but this time we set it to ***Closest Point By Surface***. This will read a point closest to a particle (as in: right under the particle) on the surface. Check ***Animated Surface*** and ***Geometry Save Mode*** as well here!

As an output we can select an ***Output Standard*** Sub-operator this time. Set it to ***Vertex Color Channel***. This will write the indicated point color into the vertex color channel that we then feed back into our flow.



You can rename the Data Operator "*Inherit VC*" or something distinguishable.

Now create a new ***Material Dynamic*** operator right under the DataOpertor and create a Material with a ***Vertex Color Map*** in the ***Diffuse Channel***. The Vertex Color from the Data Operator will be written dynamically into this map and your material will always automatically update if there is shader changes on the object you are reading data from!



When you re-cache your system now and render you should have similar results as with the Mapping Object operator but learned a few new tricks with *ADM* :)

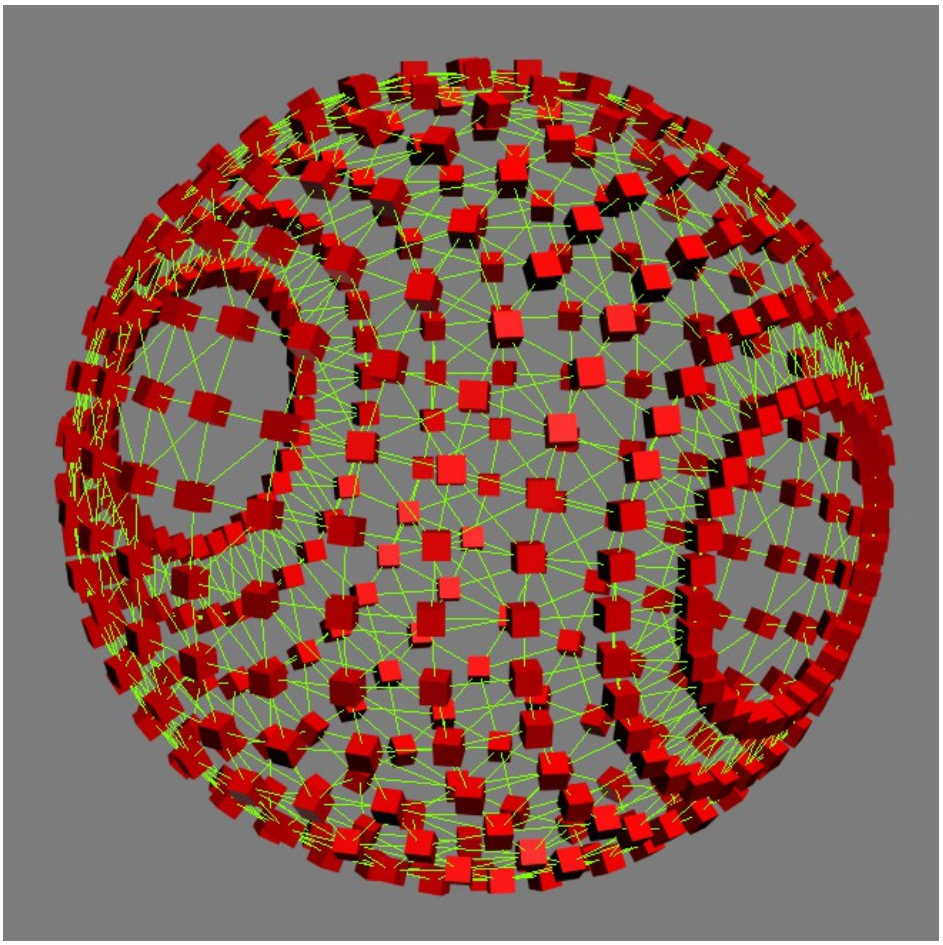
For the Black Eyed Peas music video we used this exact technique. Just with match moved characters that followed the talent plate closely. We then camera projected the plate onto those match moved Biped rigs using ***Camera Map Per Pixel*** map in the diffuse channel on a material.

Breaking the Blimp

Break compound structures using mParticles

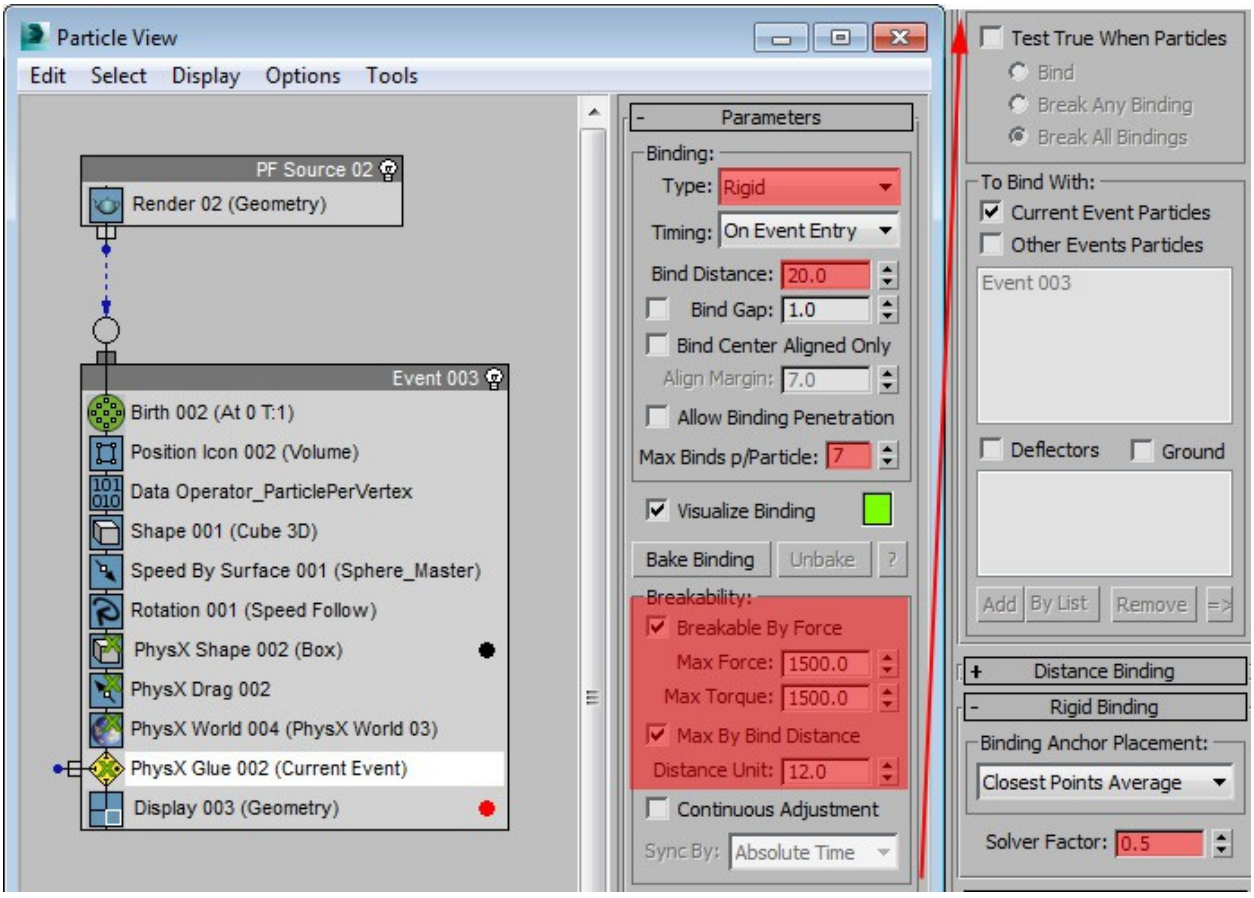
mParticles Glue binds particles to all kinds of structures from ropes, cloth over metal & glass.

The secret to speed when using mParticles is to use non intersecting proxy particles and then glue hires meshes to them. In our case the effect can be sold with just boxes that interact as our rigid bodies and binding compounds.



To allow for nice massive metal like rigidity we need 3 key components: 1) high enough ***binding count*** with neighboring particles, 2) a bit of ***drag*** to keep things in check and lastly 3) low ***time scale*** to sell the mass of it.

In order to break a binding structure you need to make up your mind about the binding qualities first. mParticles have 3 binding types for various use cases. For a rigid metal structure using the ***Rigid*** binding type makes the most sense. It has the ability to break bindings bases on Force and Torque. It's overall stiffness can be controlled by ***time sub samples*** and the ***Solver Factor.***

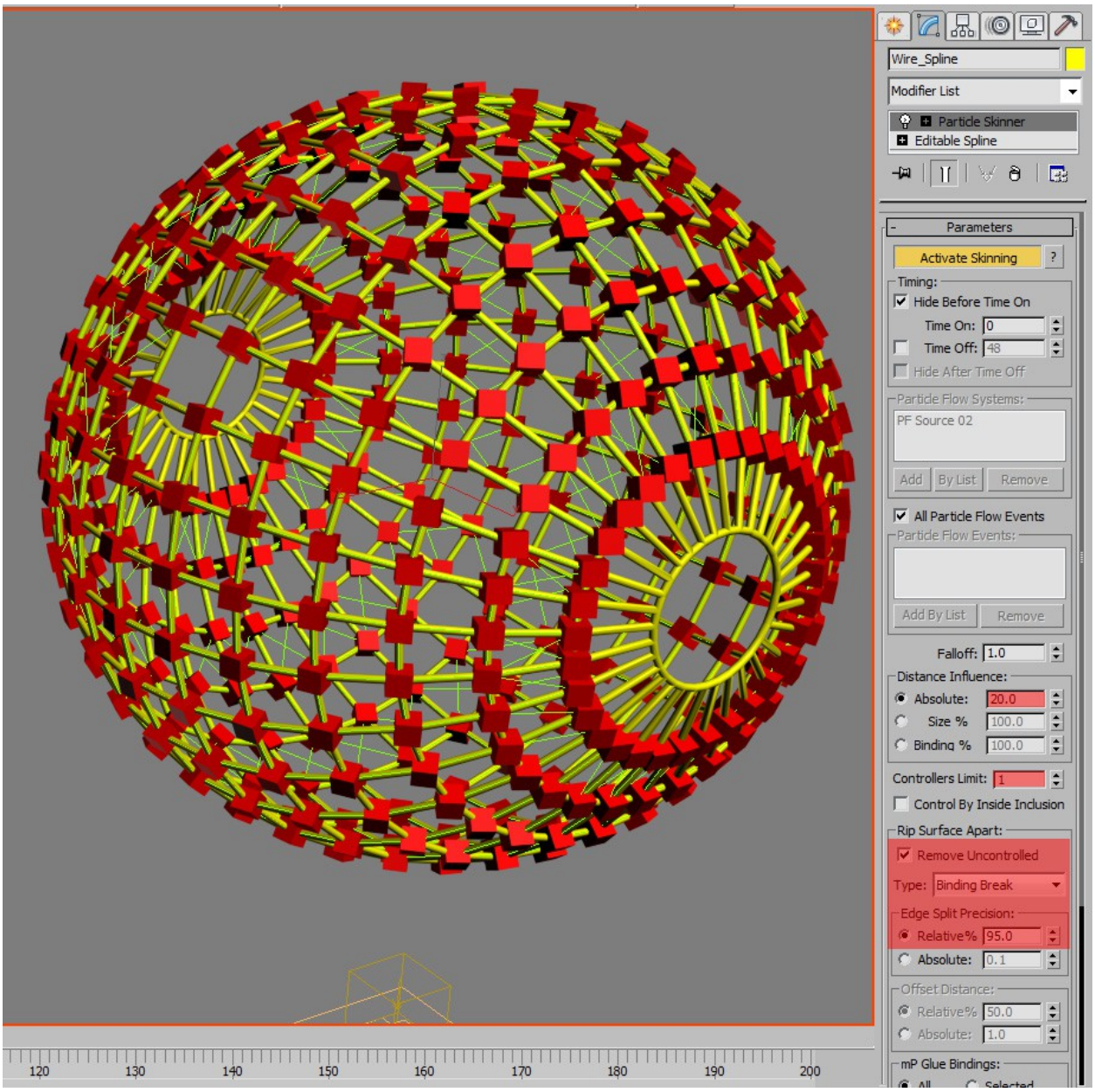


Important! mParticles Tests have to be located ***UNDER*** the World operator.

For bindings to happen their bind minimum distance should exceed the distance between two particles. More bindings = stiffer binding behavior.

For an interesting breaking pattern we activate ***Max By Bind Distance***. This will allow for stronger bindings between closer particles within the given Distance Unit radius. This can result in patches of particles breaking away while internally stating connected.

After the motion of the particles feel right it's time to skin the skeleton and chassis to the particle system. The new mParticle Skinner does just that. Think of it like a SkinWrap modifier but for all things particles.



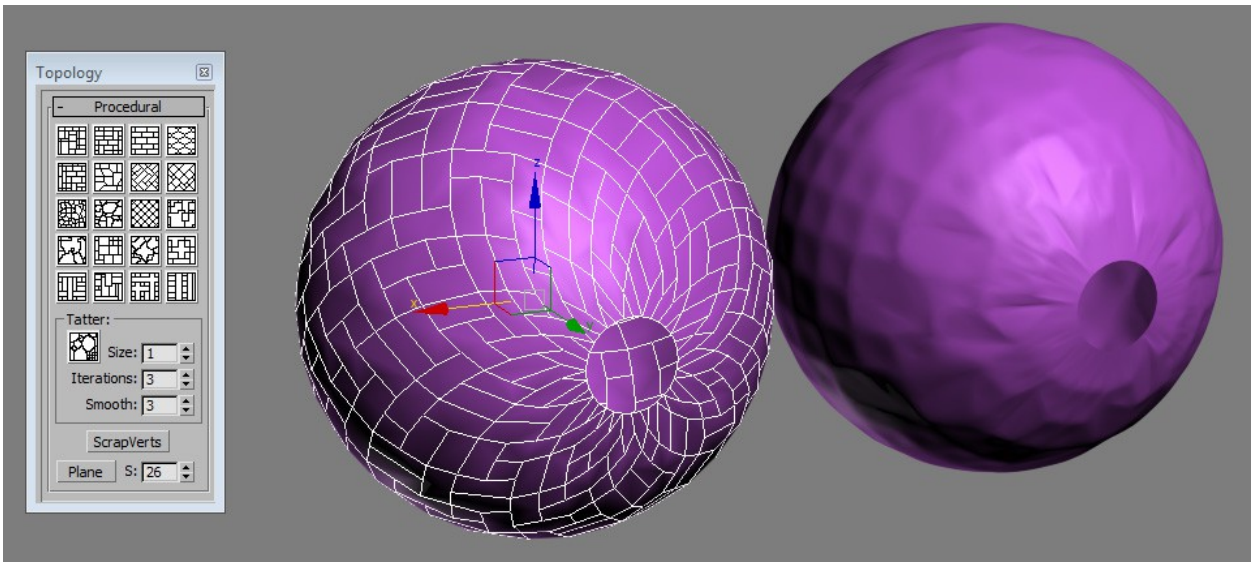
After applying the modifier add in the particle system and hit activate skinning. With the default settings applied we will get some kind of melting effect on the skeleton. First we need to adjust the ***Distance Influence*** and ***Controller Limit***. We only want 1 particle at a time to control a certain area which is defined in the Distance Influence. This distance should be close to our binding distance.

Now our metal bends and twists but doesn't break yet. We need to set the ***Rip Surface Apart*** settings from None to ***Binding Beak,*** also check ***Remove Uncontrolled***.

If you want to render with Vray and motion blur e.g. you might want to adjust your ***Sustain Topology*** settings. There is a rule of thumb to go by: MBlurDuration*4800/FPS + 1

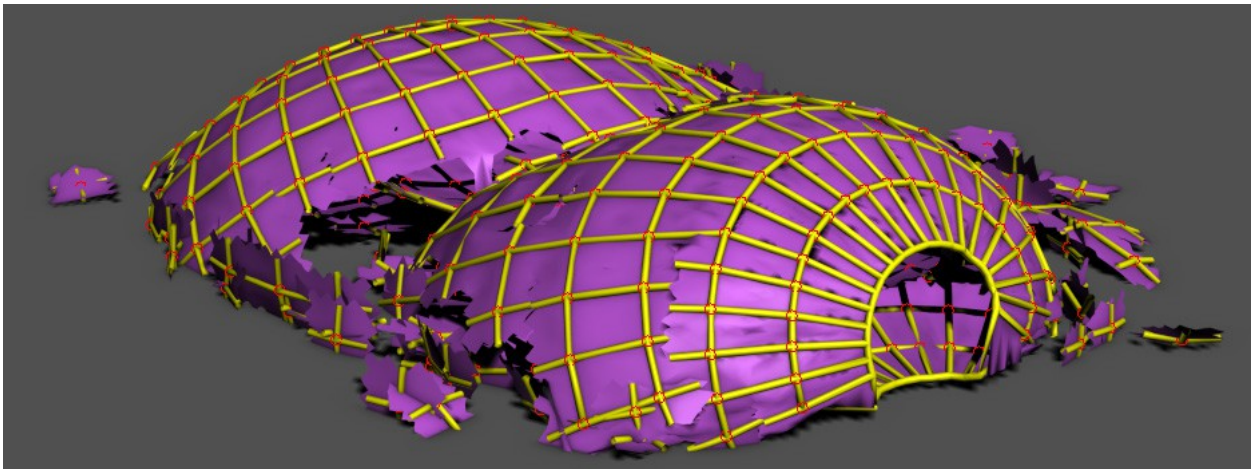
4800 is the amount of ticks per second in Max.

For the cloth part of the blimp we can generate a more interesting pattern with the ***Graphite tools*** and a ***Tessellate*** modifier, eventually a ***Relax*** modifier too if normals get busted:



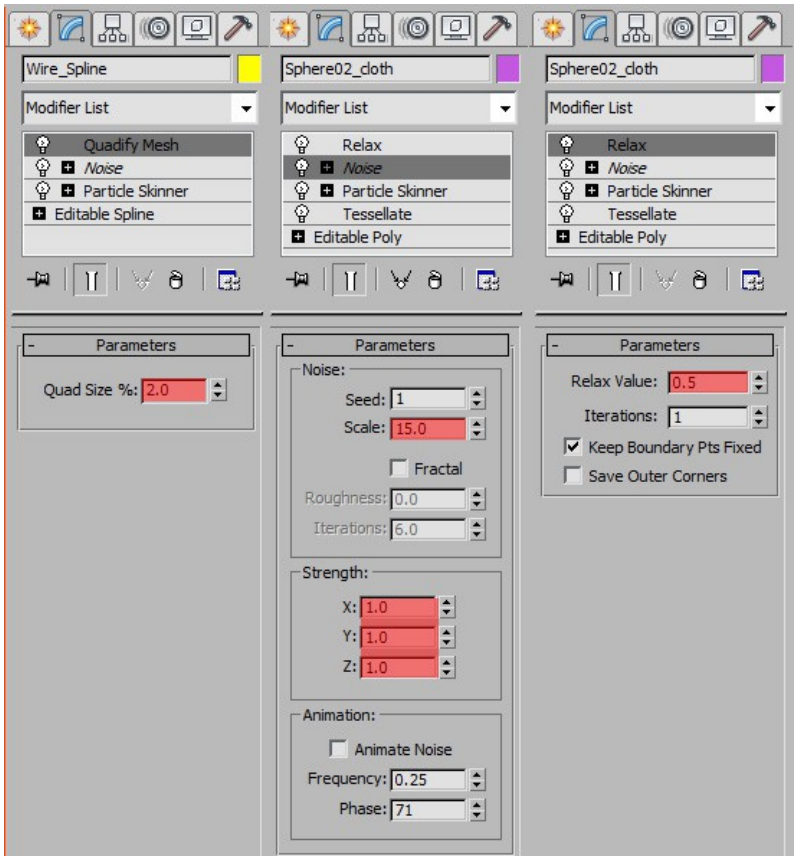
After you are happy with the overall detail of your cloth patches it's time to apply the Particle Skinner modifier here as well. Apply the same settings as you did for the skeleton structure.

Hope fully your result should look something like this:



As the final step we can touch up the effect by adding additional modifiers on top of the Particle Skinner.

A ***Shell*** modifier to the cloth to give it width. A ***Noise*** modifier to give both the skeleton and the cloth subtle secondary motion. A ***Quadify*** modifier on the skeleton to give it more precision on the crack edges.

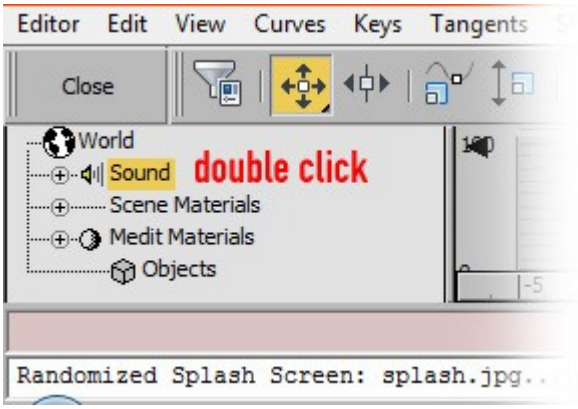


Sounds good! ADM (Advanced Data Manipulation) for sound manipulation

Use helpers and sound to drive your particle simulation

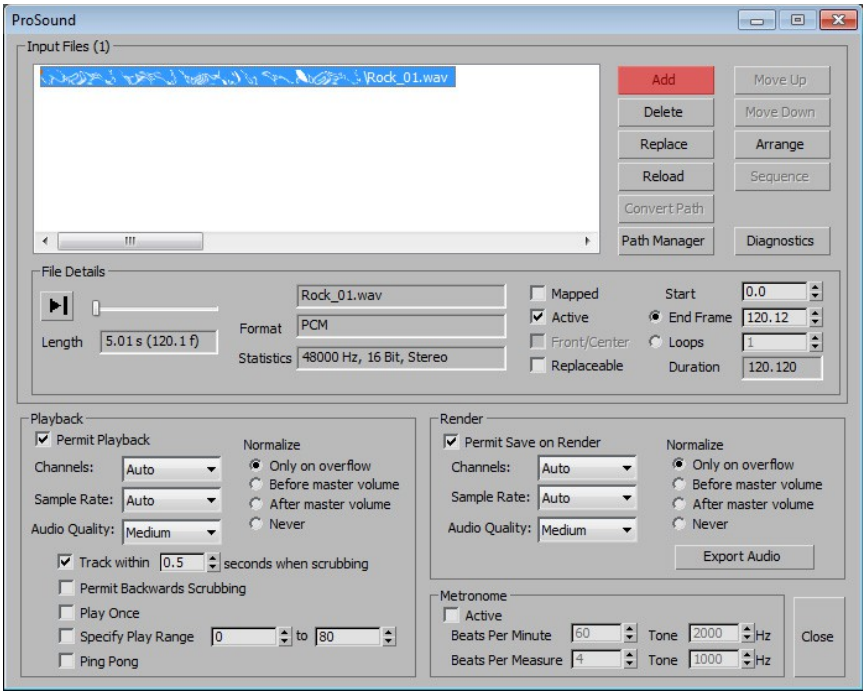
ADM adds a great range of possibilities to PFlow being virtually a node based code toolbox.

With ADM you not only have control over particles. You can also read and write values to scene objects. We want to scale a particles as boxes based on sound waves in this tutorial. We are going to utilize the ***Object Parameter*** and ***Shape Control*** sub-operators in order to do this. One reads values from scene objects or any spinner on every modifier in Max, one influences objects in the scene based on values set inside a Data Operator.

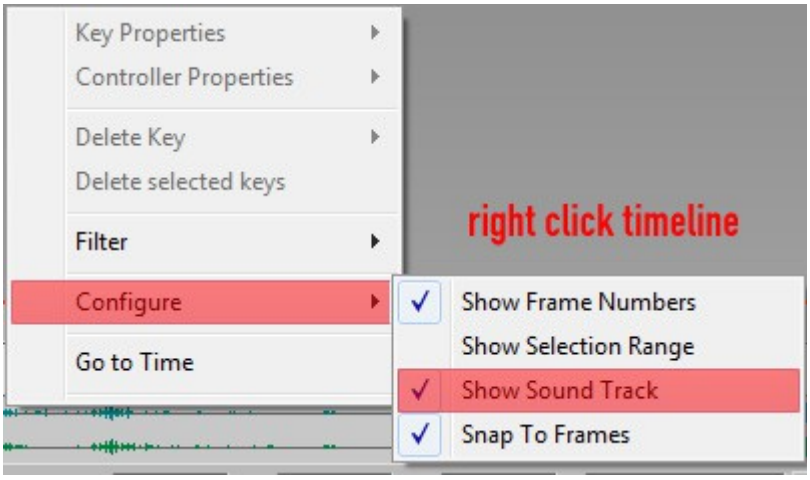


But before we dig in we should prep our scene to play back sound and visualize it's wave forms.

Open the ***Curve Editor*** and double click on ***Sound***. Select the provided WAV file and make sure Permit Playback is checked.

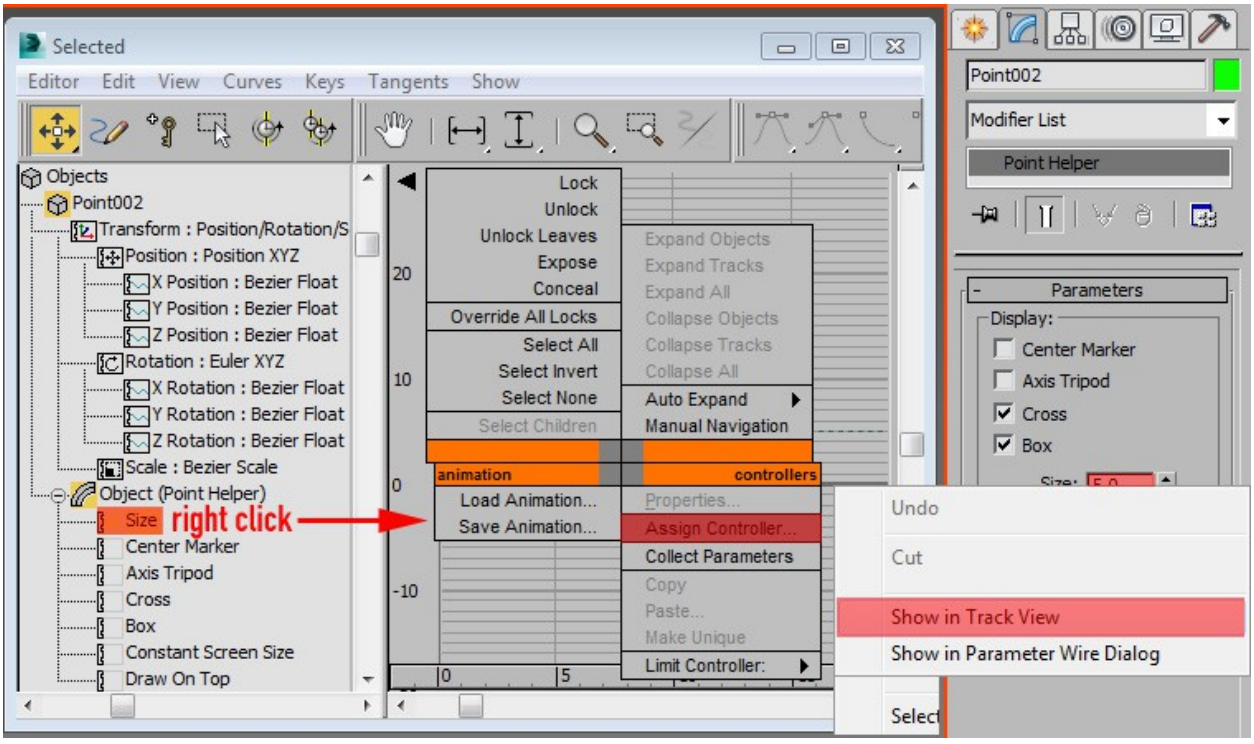


To see the sound waves in the time line right click and check ***Configure > Show Sound Track***

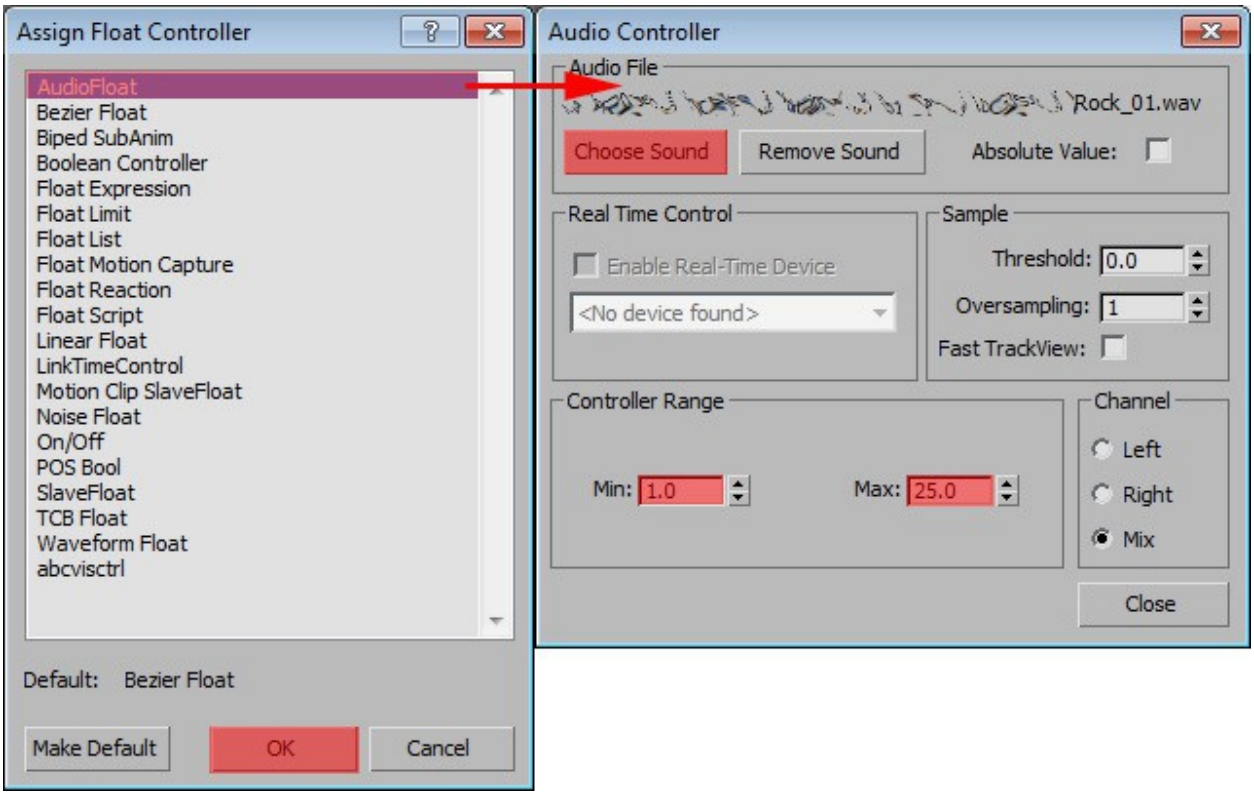


Now you should hear and see the WAV file when scrubbing the time line.

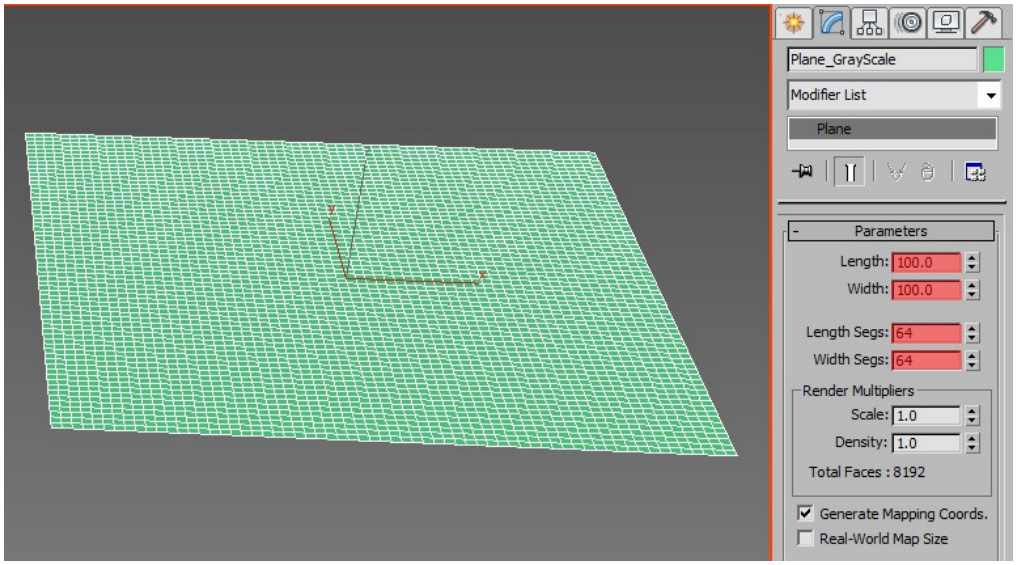
Next we create the object that is controlled by sound that then controls the particles. Create a ***Point Helper*** and right click on the ***Size*** spinner. Select ***Show in Track View*** and right click on the ***Size*** inside the Track View to bring up the Controler Menu. Select ***Assign Controller***.

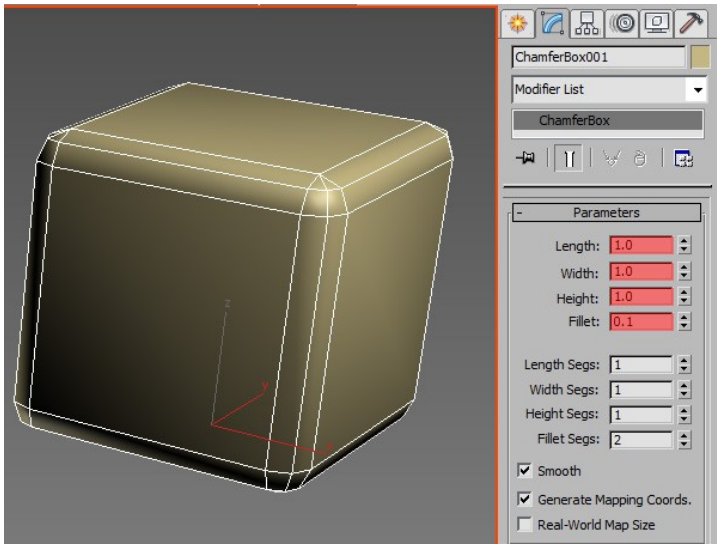


We are going to Add an ***AudioFloat***. Choose the same sound as in the timeline and adjust the min and max ***Controller Ranges***. Don't worry too much about the values we can always go back and adjust those or multiply the value inside of ADM!



The last step before we finally create our particle system is a distribution plane and shape. Create a ***Plane*** with ***64x64 Segments*** and a desired size.

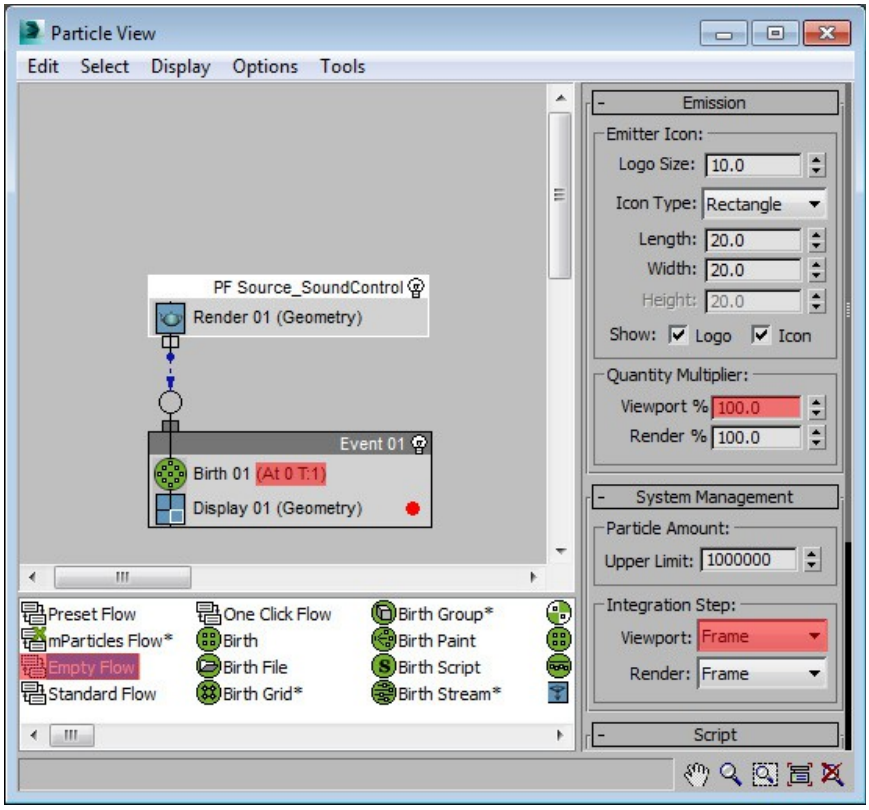




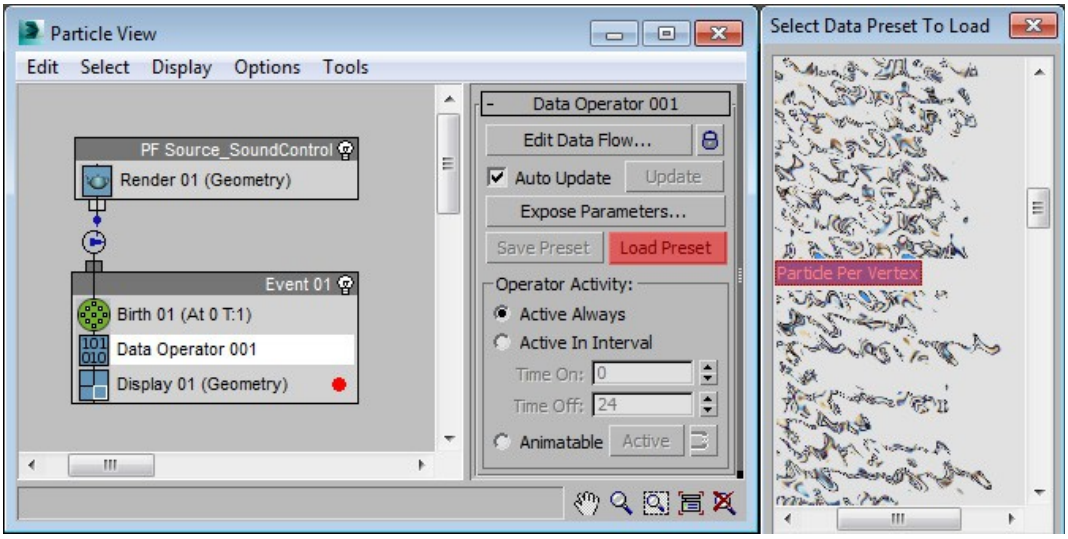
Also create a ***ChamferBox*** with just the size of 1 in all dimensions and a slight Fillet.

Now all preparation for our system is done. Create an ***Empty Flow*** and make sure you see ***100%*** of the particles in the Viewport and that the sub-sampling is set to ***Frame***. We don't need a lot of sub-steps unless we have fast metal or techno.

Spawn just one particle on frame 0 only.



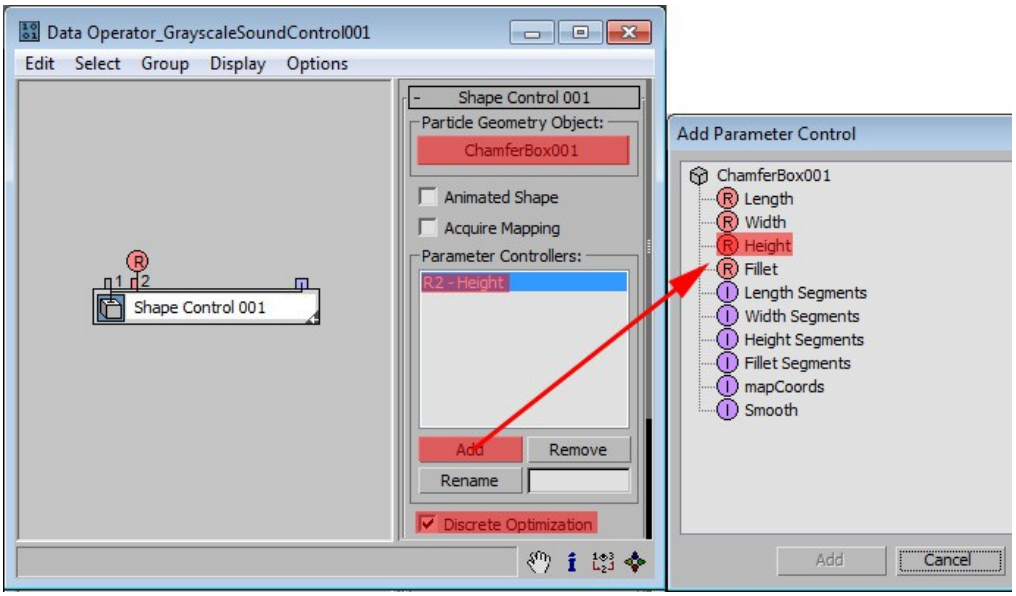
Now create a ***Data Operator*** and load the preset called ***Particle Per Vertex.***



This is a preset that ships with ADM and is thus not further discussed. It spawns particles per vertex on a selected object. Once you pick your distribution plane you should see all particles sitting on a vertex each with *no* overlapping (hence the Data Operator over the use of a Position Object! Alternatively you could use a Birth Grid with 0 height as well).

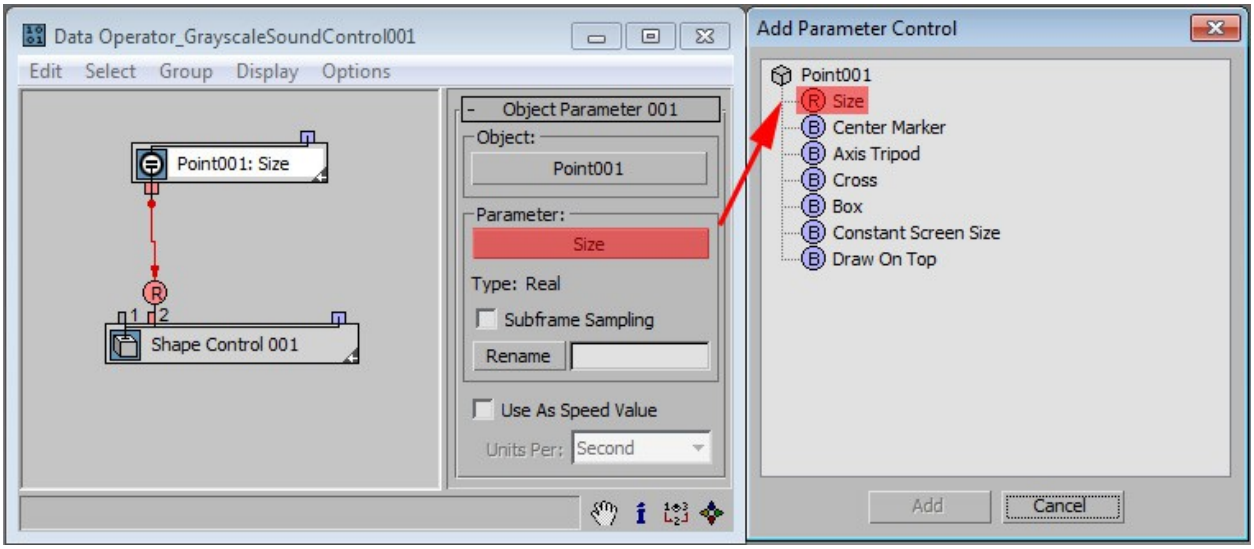
And now finally we can create the part where the magic happens and it all comes together.

Create another ***Data Operator.*** This time we create a ***Shape Control*** sub-operator.

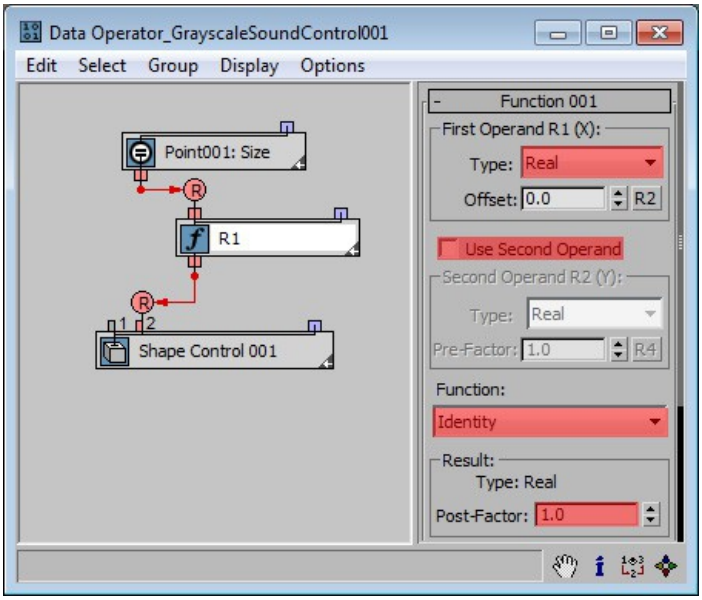


Select the ChamferBox as ***Particle Geometry Object*** and as the controlled parameter ***Add*** it's ***Height***. Check ***Discrete Optimization***, that will round fractures to full values like 1,2,3,etc.

Now our ChamferBox is a particle! If you set the Display type to Geometry you will see them all sitting on the distribution object. Now we can pull all kinds of tricks on this box with ADM. Next create a ***Object Parameter*** sub-operator and pick the ***Point Helper*** you created earlier with the sound influencing it's size. Pick the ***Size*** as Parameter to read values from.



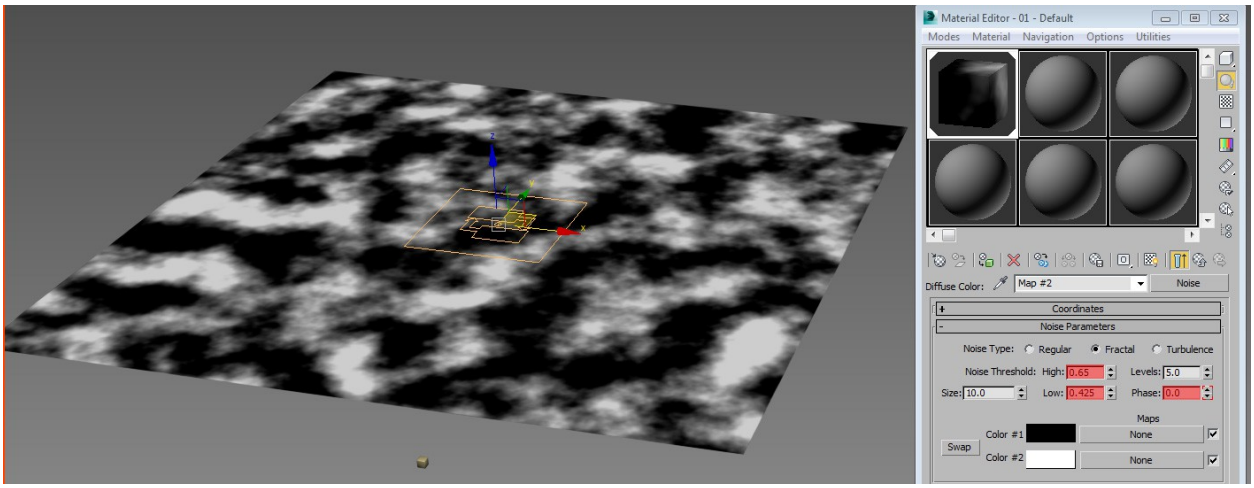
Now the sound is controlling the Height of our ChamferBox via the size of the Point Helper controlled by ADM. In order to be able to multiply the effect you can add in a ***Function*** Sub-operator:



Make sure to ***uncheck*** the ***Second Operand*** and set it to ***Real*** to comply with the values put out by the Object Parameter sub-operator. Set it's Function to Identity to act just as a pipe through. The multiplication of th effect is achieved by the ***Post-Factor.***

If you make a preview now the particles bump up and down to the beat of the sound. But wouldn't it be nice to have a oscilloscope style amplitude? Let's build that!

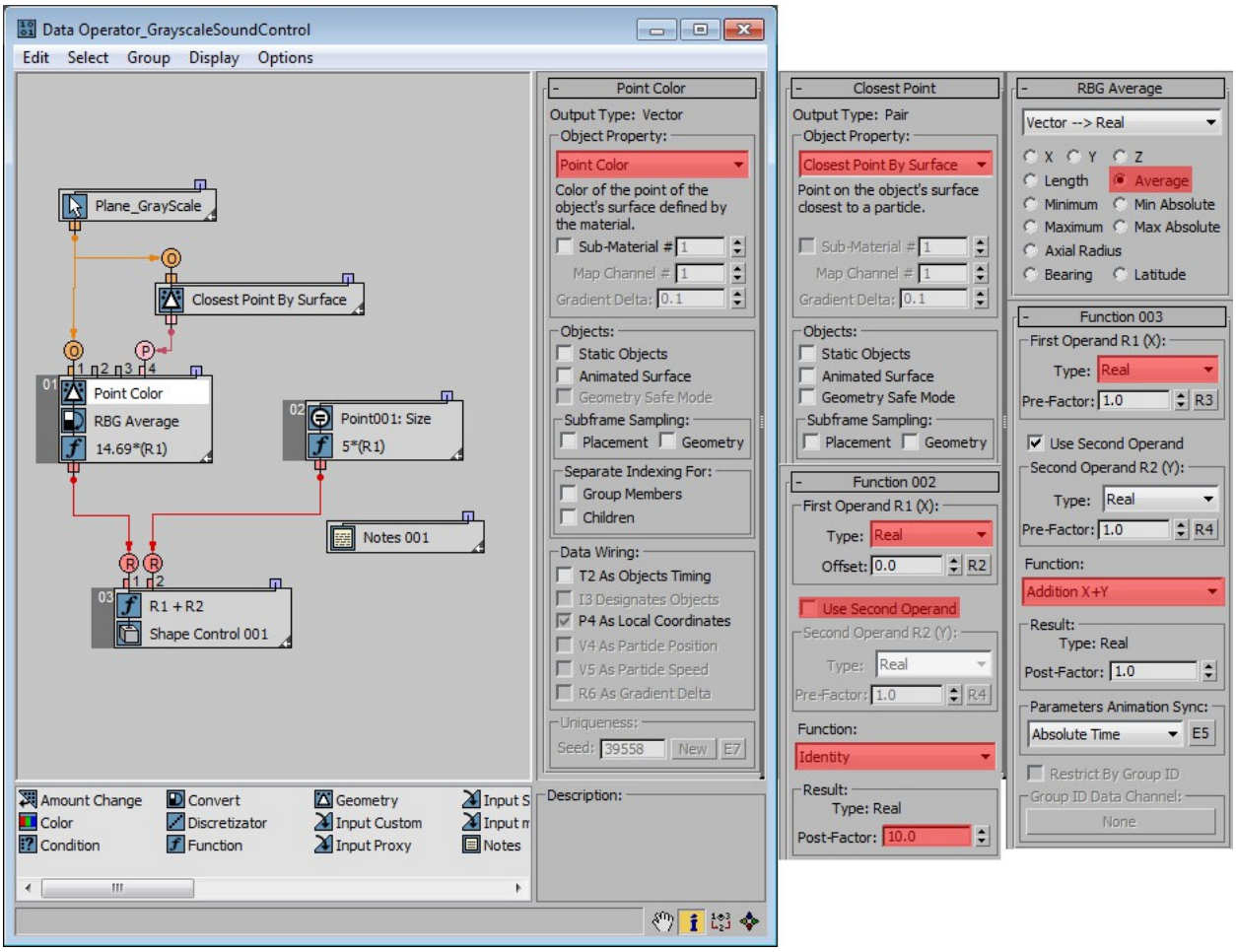
First we need to assign a texture map to our distribution plane. I went with a ***Noise Map*** because it has nice high and low variations in the shading. Adjust the Type to ***Fractal*** and make the ***Low*** and ***High*** values sharper. Then animate the noise ***Phase*** from 0-10 over the duration of the scene. This will crate an interesting crawling noise pattern. Let's merge that with the sound height control.



We add a ***Select Object*** sub-operator to the Data Operator and pick our plane, again as we want to read data from a scene object and we need two ***Geometry*** sub-operators, we are reading data from a piece of geometry. We want to read the ***Point Color*** so pick that from the drop down menu. We don't need sub-frame sampling or Animated Surface checked as out plane is static. Point Color demands two inputs: *WHICH* object to read color from and from *WHERE* on the object. So we need another value here. To define where on the object it needs the other ***Geometry*** sub-operator but this time we set it to ***Closest Point By Surface***. This will read a point closest to a particle (as in: right under the particle) on the surface.

Now that we have defined that color as a Vector output by default we need to connect it to the Shape Control sub-operator. In order to do that we create a new ***Function*** sub-operator, set it to ***REAL*** values and make sure it is set to ***Addition***. Drag the Object Parameter tree into the second operand of the function and the vector from the Point Color into the first. ADM is smart enough to automatically add a ***Convert*** sub-operator to make real values out of the vectors.

If you want to control the influence of the gray scale map on the plane you can even add in another ***Function*** right after the Convert sub-operator that you set up just like the multiplier Function under the Object Parameter. Just use one operand and set it to ***Identity*** to act as a pipe through. The ***Post Factor*** then acts like a multiplier.



Your end result should look something like this. Make a preview: