Water Flow for UDK

Dmitry Barannik
redcatbox@gmail.com

Apr, 2013

Table of contents

Purpose

Any developer wants to use extraordinary effects in his projects, and we always seek for new features and solutions.
Purpose of this tutorial – make full and complete description of a way to construct “Water Flow” algorithm for UDK users.

I used few cool documents

by **Valve**
Water Flow in Portal 2
Making and Using Non-Standard Textures

and **Naughty Dog**
Water Technology of Uncharted
The Tricks Up Our Sleeves

and I really happy that professionals share their knowledge with simple people.

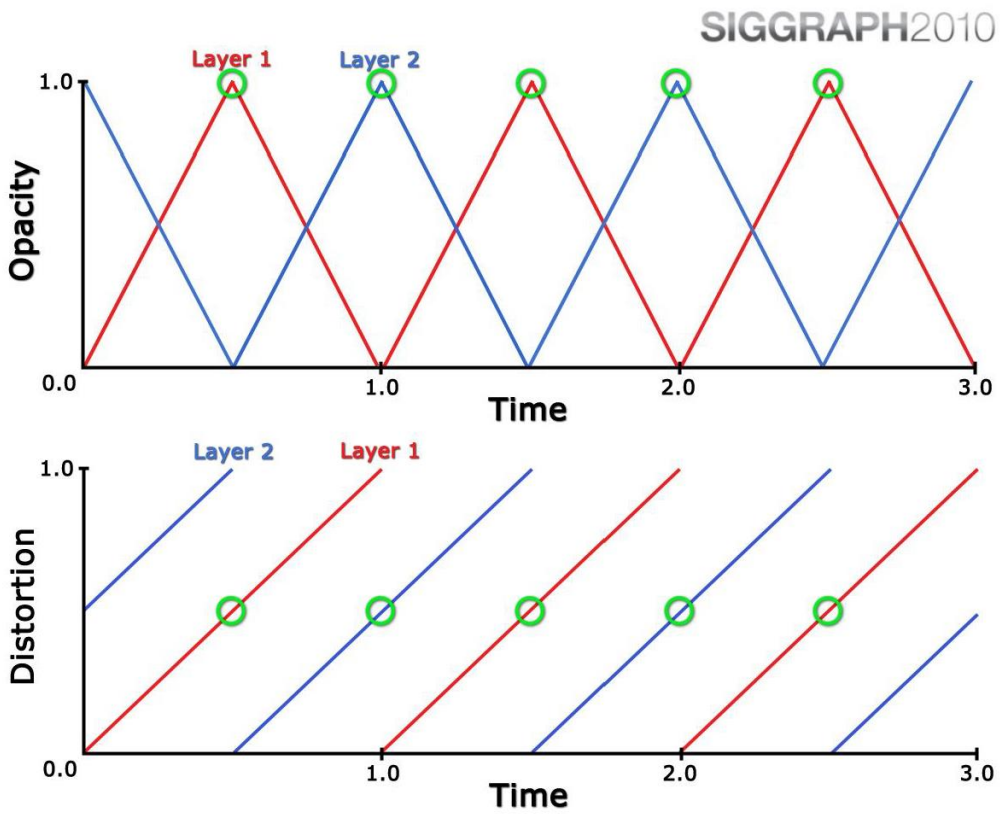
Materials setup

Flowing normal maps

Simply repeat page from Alex Vlachos document.

Flowing Normals

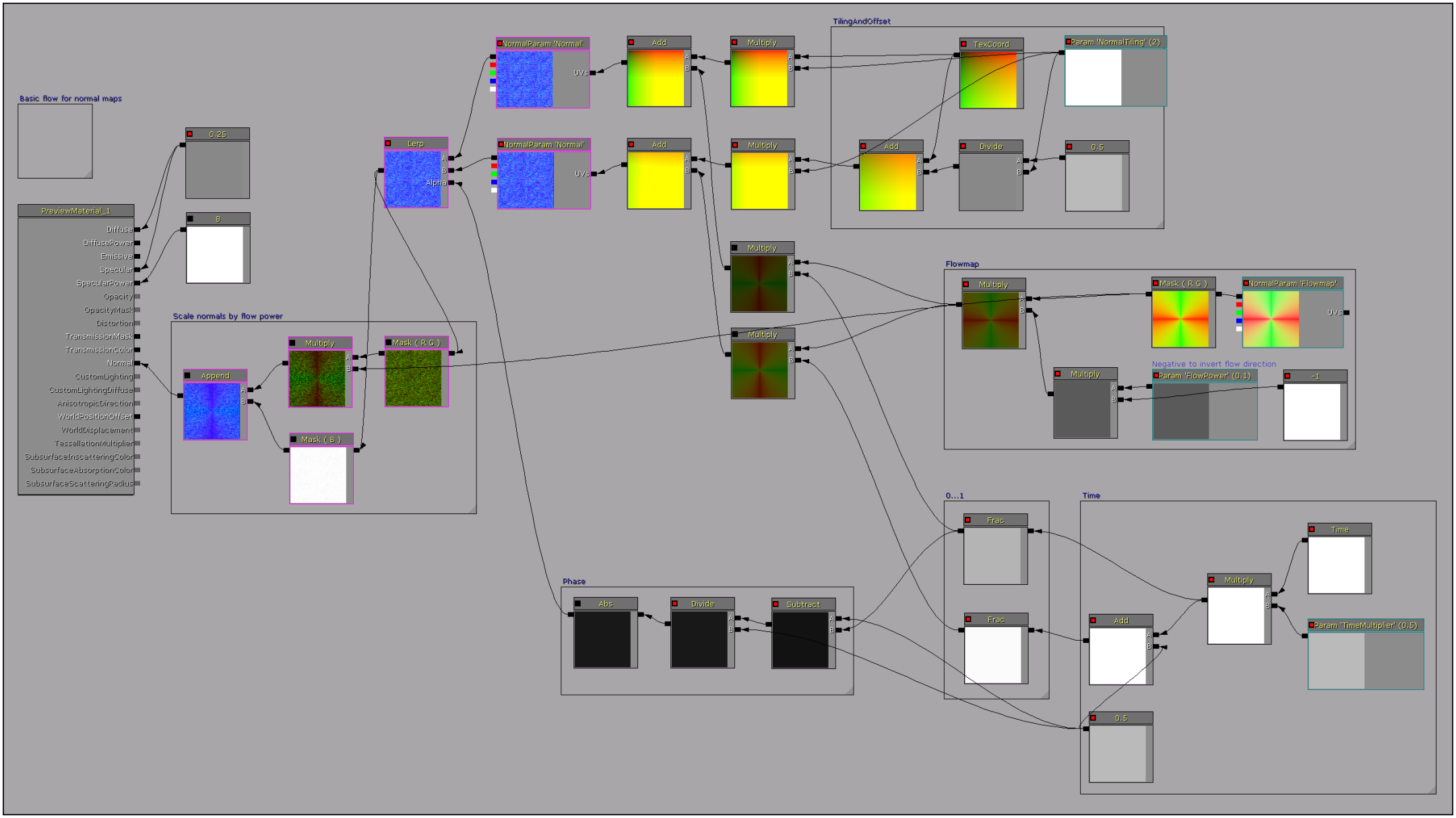
- Flowing normals would repeat an interval from zero to some fraction with the peak (center) of the interval at half distortion



To create flow, you need to distort texture's UVs with time, and cycle this process for 2 textures, which will replace each other.

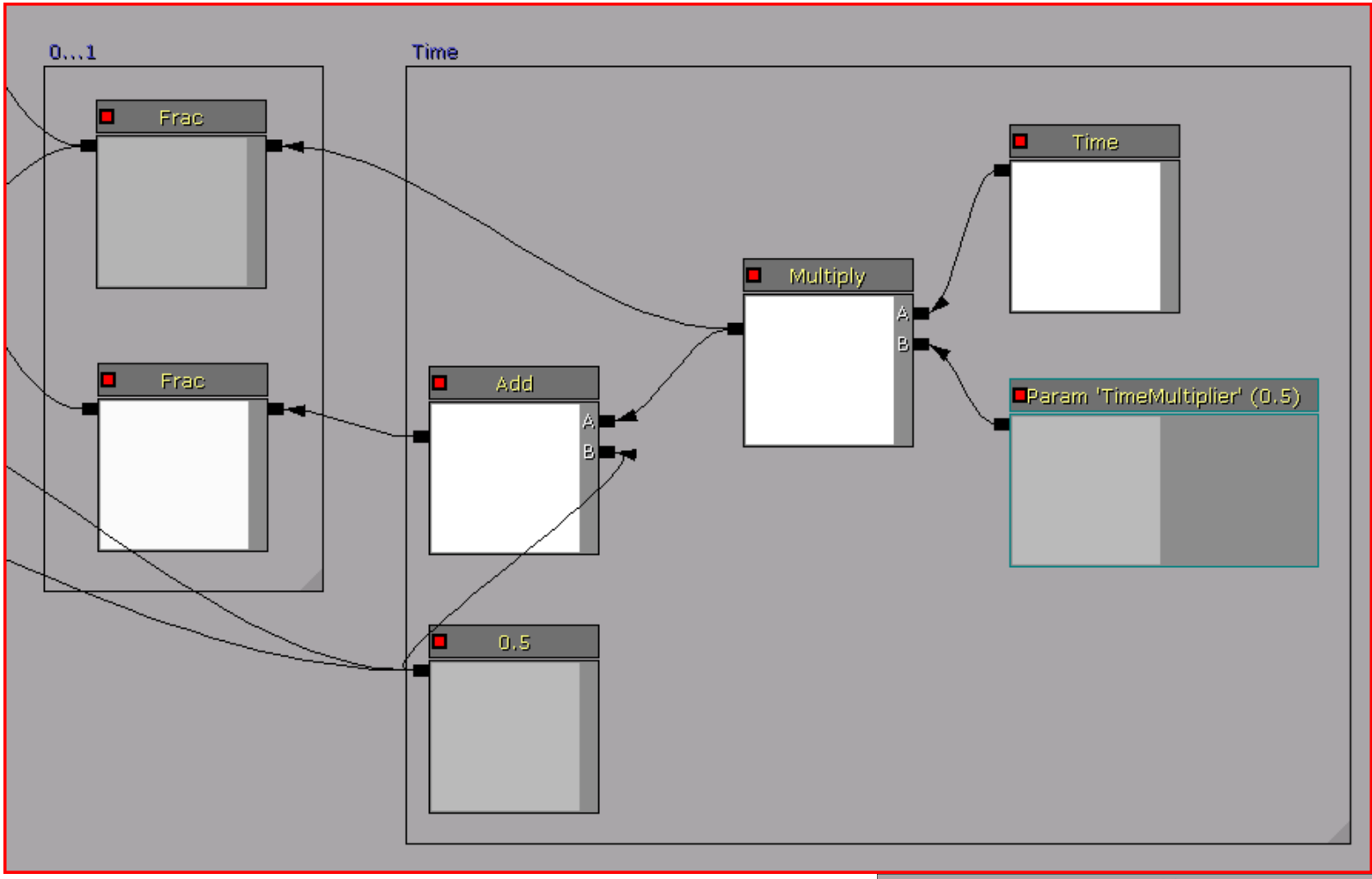
Basic flow for normal maps

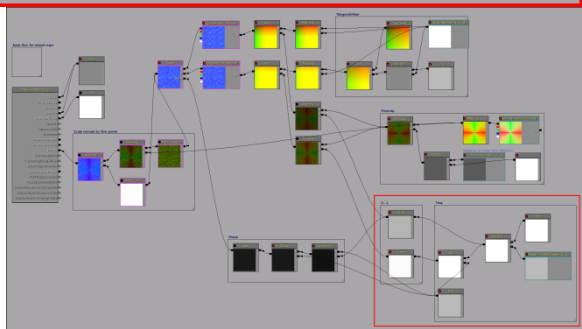
Material layout:



For normal maps, distortion range should be 0...1. So, you need simple saw-tooth function.
Two Time inputs:

```
Frac(Time*Mult)
Frac(Time*Mult + 0.5)
```



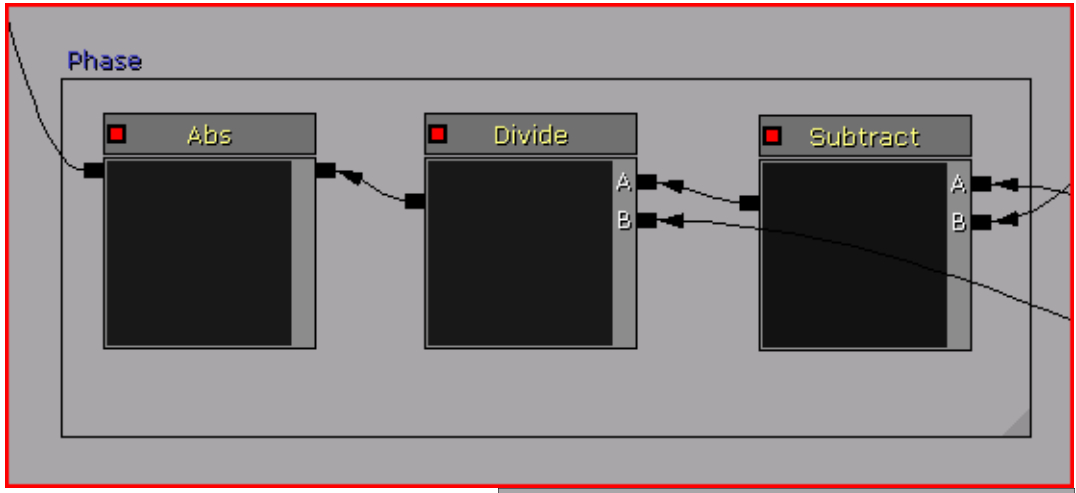


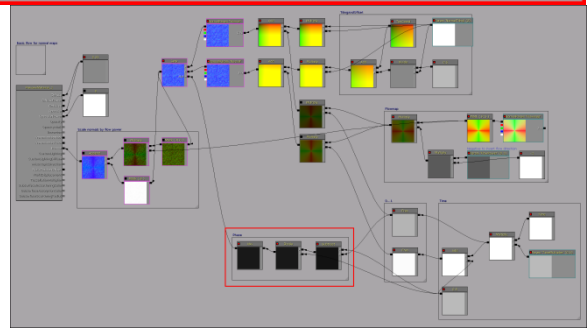
Phase, which will be alpha value for Lerp of 2 normal maps.

```
Abs(0.5 - Frac(Time*Mult))/0.5
```

Phase simply changes from 1 to 0 and again to 1:

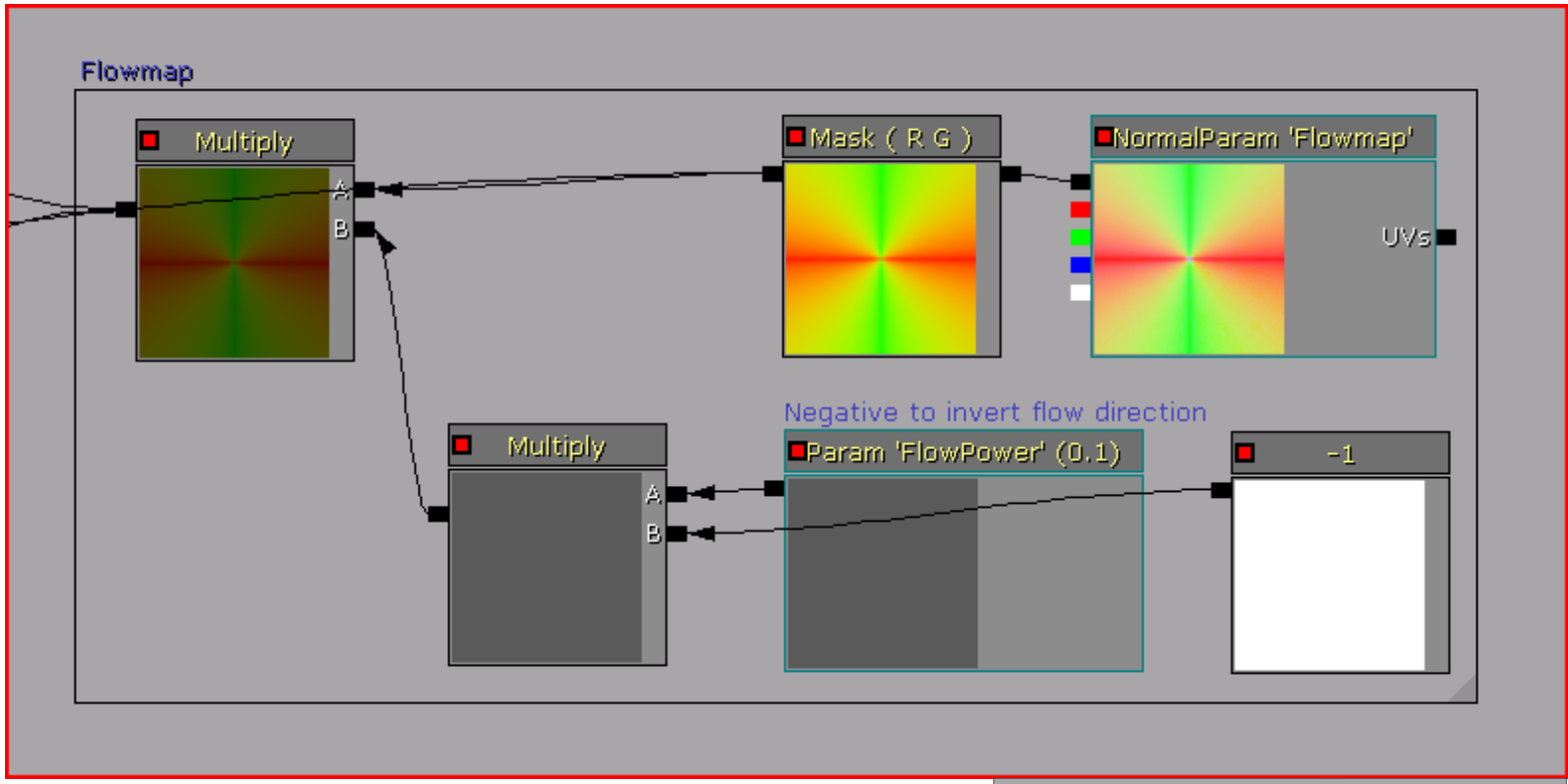
Time = 0,	Phase = 1,	2nd map is fully visible,
Time = 0.5,	Phase = 0,	1st map is fully visible,
Time = 1,	Phase = 1,	2nd map is fully visible.





Flow map, multiplied on some scalar, which represents "power" or "speed" of distortion.

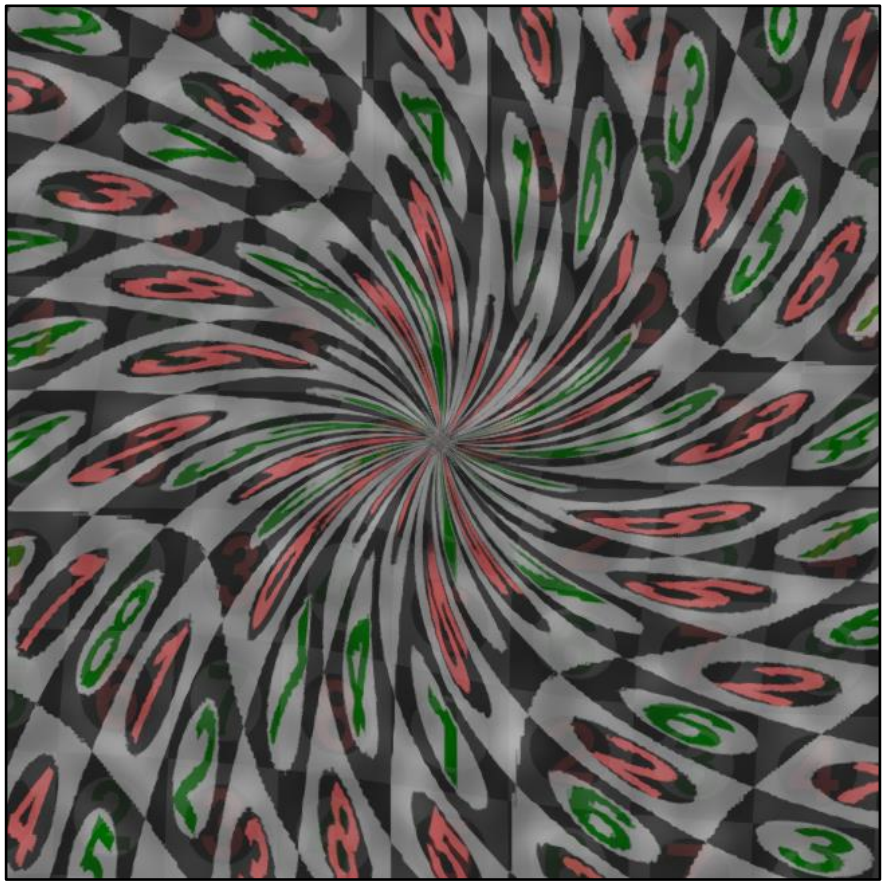
```
Flowmap.rg*(-FlowPower)
```



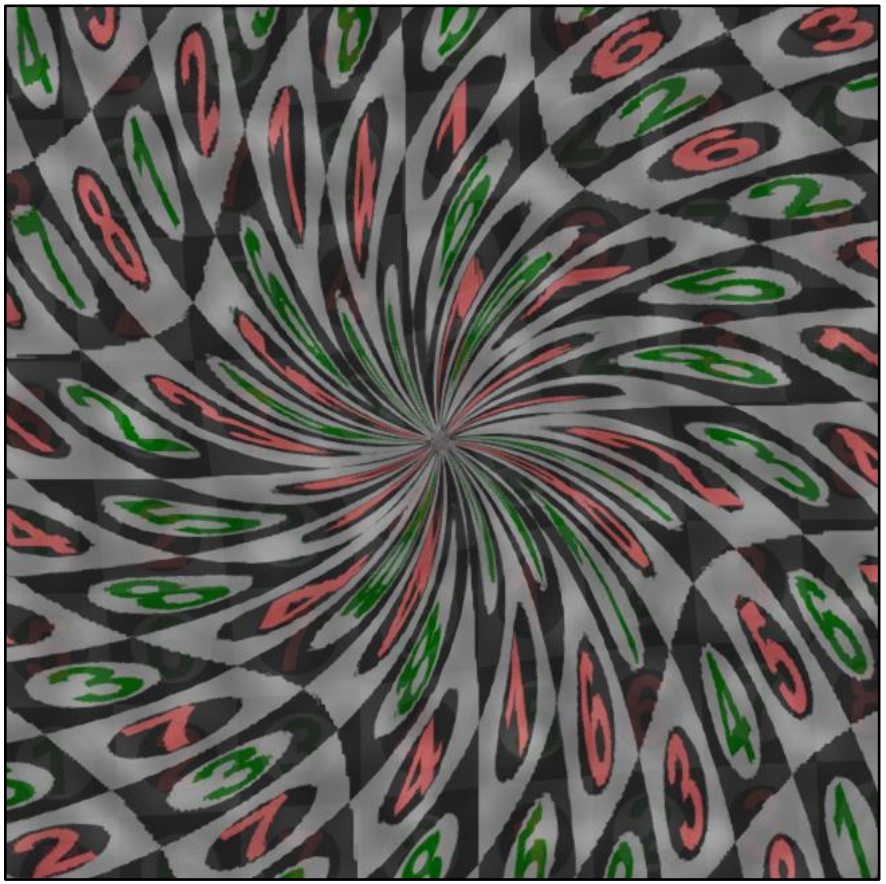
FlowPower multiplied on (-1) to get correct flow vectors work.
(!) It's because UV's needs to be distorted in opposite direction to flow.

So, by default, FlowPower positive, and after multiplying on (-1) it will be negative. And flow will work as expected.
And, if you set negative FlowPower, after multiplying on (-1) it will be positive, and flow will be inverted.

Normal flow



Inverted flow

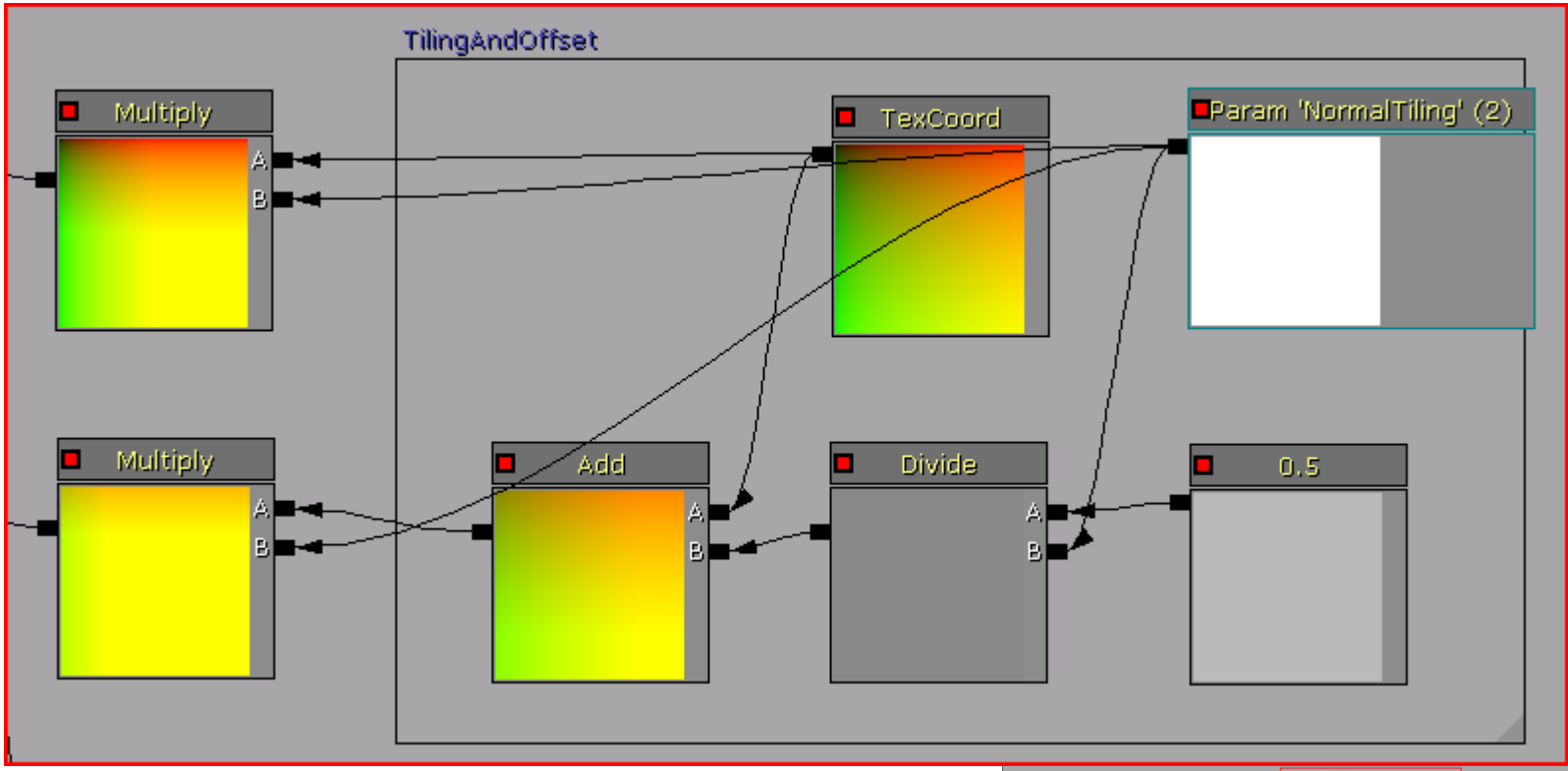


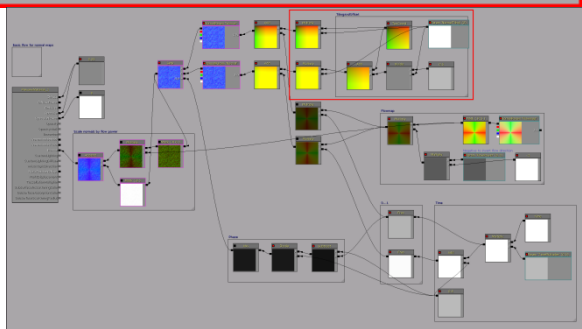
Of course you can remove that multiplier and simply use negative value by default.
I set it for usability and understanding purpose.

Tiling and offset instructions for UVs, used to break texture's repetition.

For 1st texture – `Tiling*TexCoord`
For 2nd texture – `Tiling*(TexCoord + 0.5/Tiling)`

So, 2nd texture will be shifted on 0.5 from 1st in any tiling.

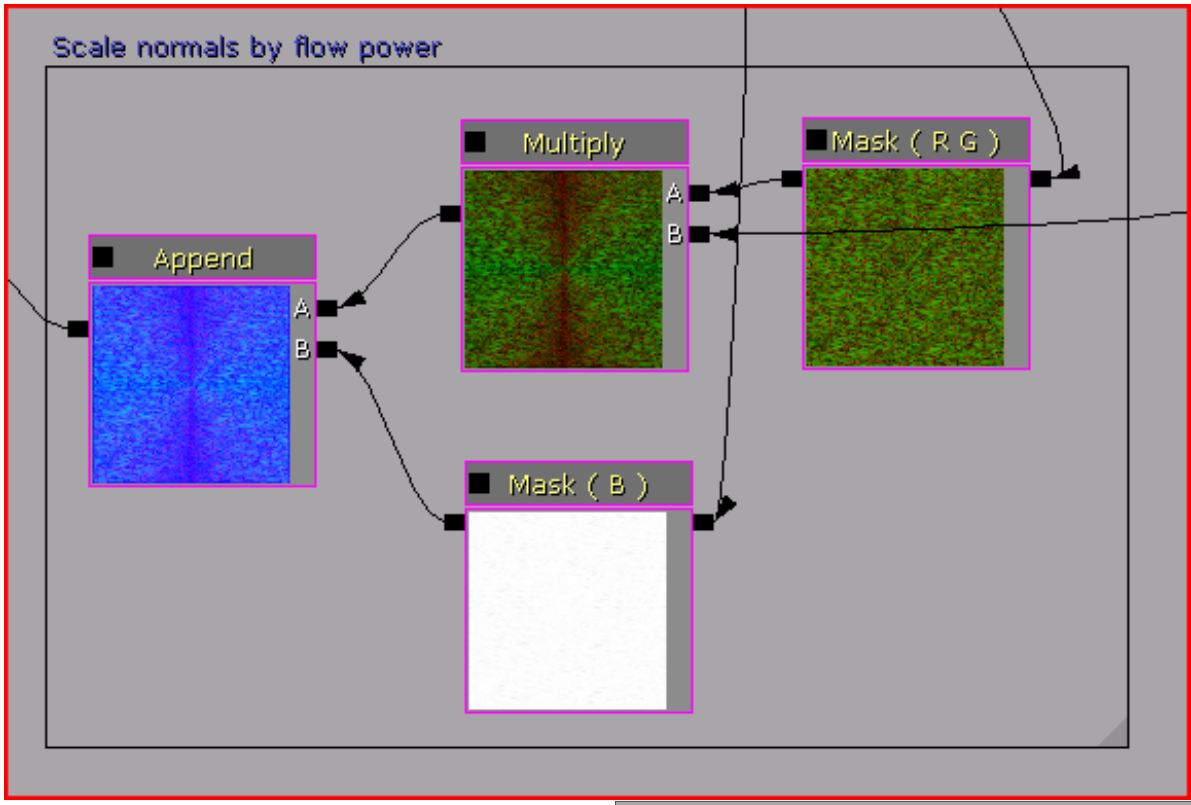


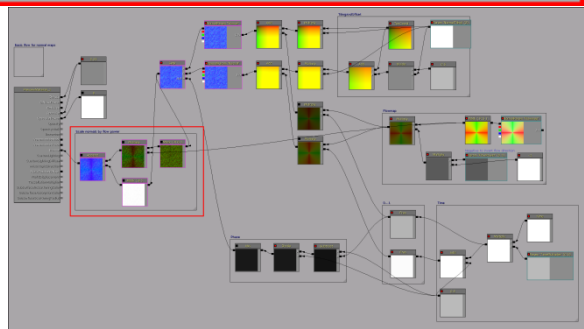


Also, you can scale normals with power of flow.

```
Append(Normal.rg*Flowmap.rg, Normal.b)
```

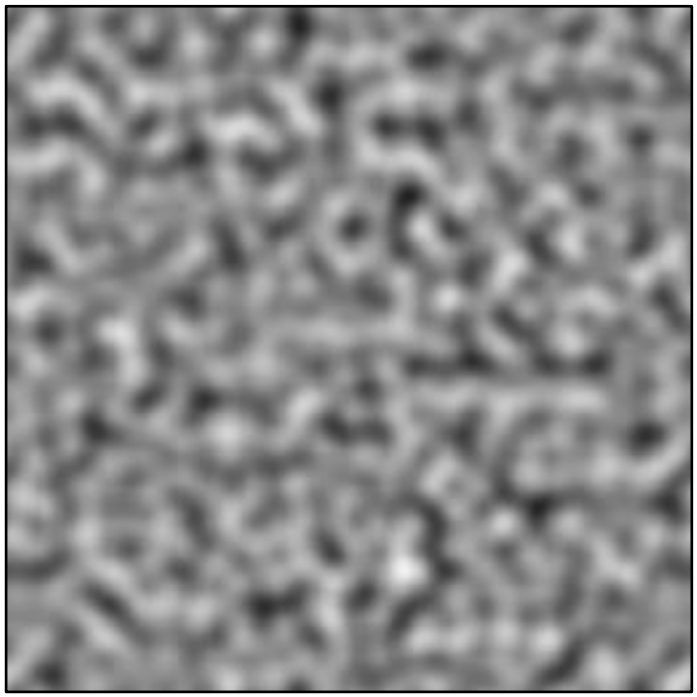
Result – flat normals where flow vectors approach to 0, and strong normals where flow vectors approach to -1/1.





Basic flow for normal maps with per-pixel phase offset

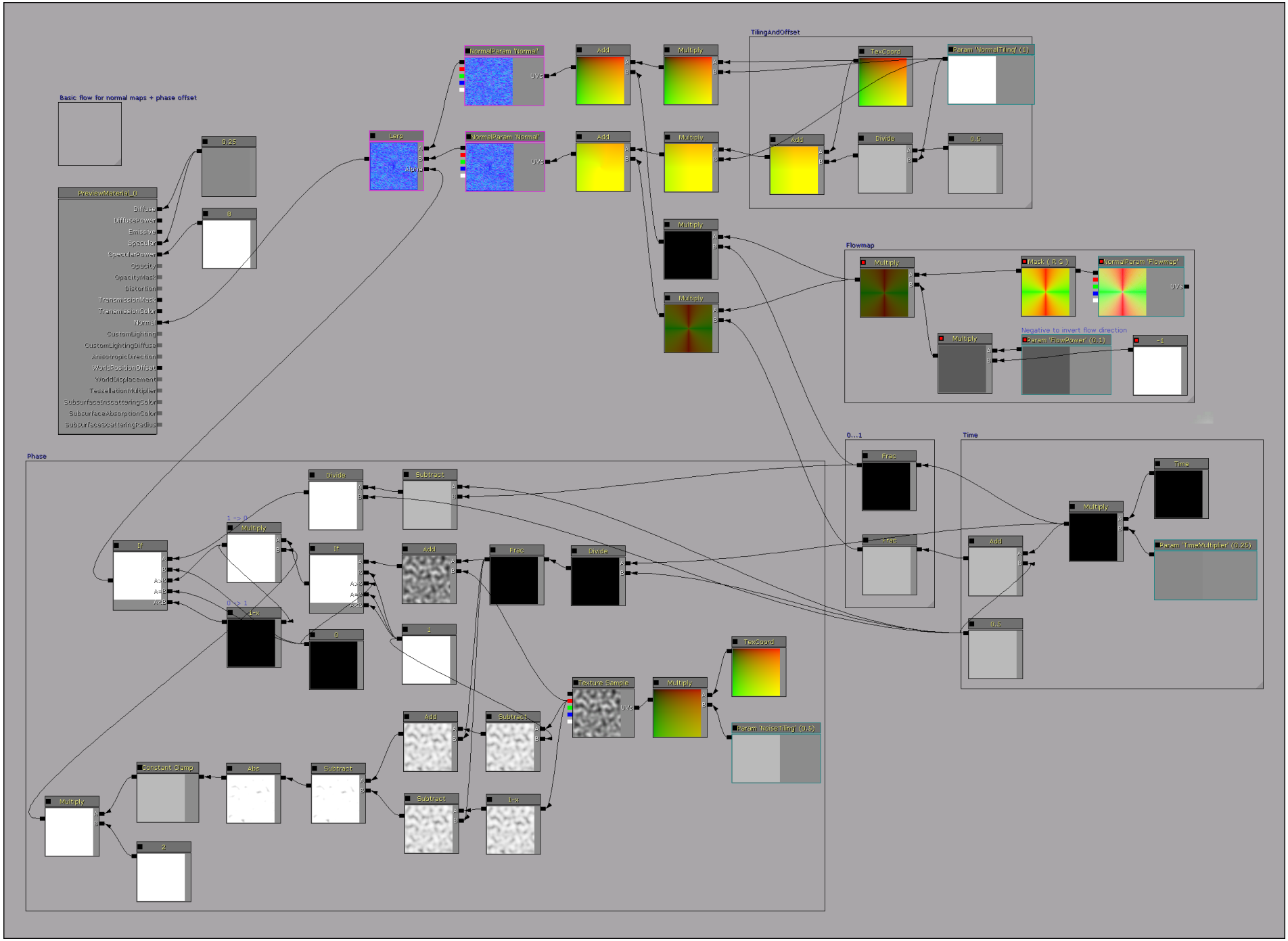
Because phase changes for whole surface, you can see effect of surface's pulsing.
You can handle this, using special noise texture, to offset phase per-pixel.

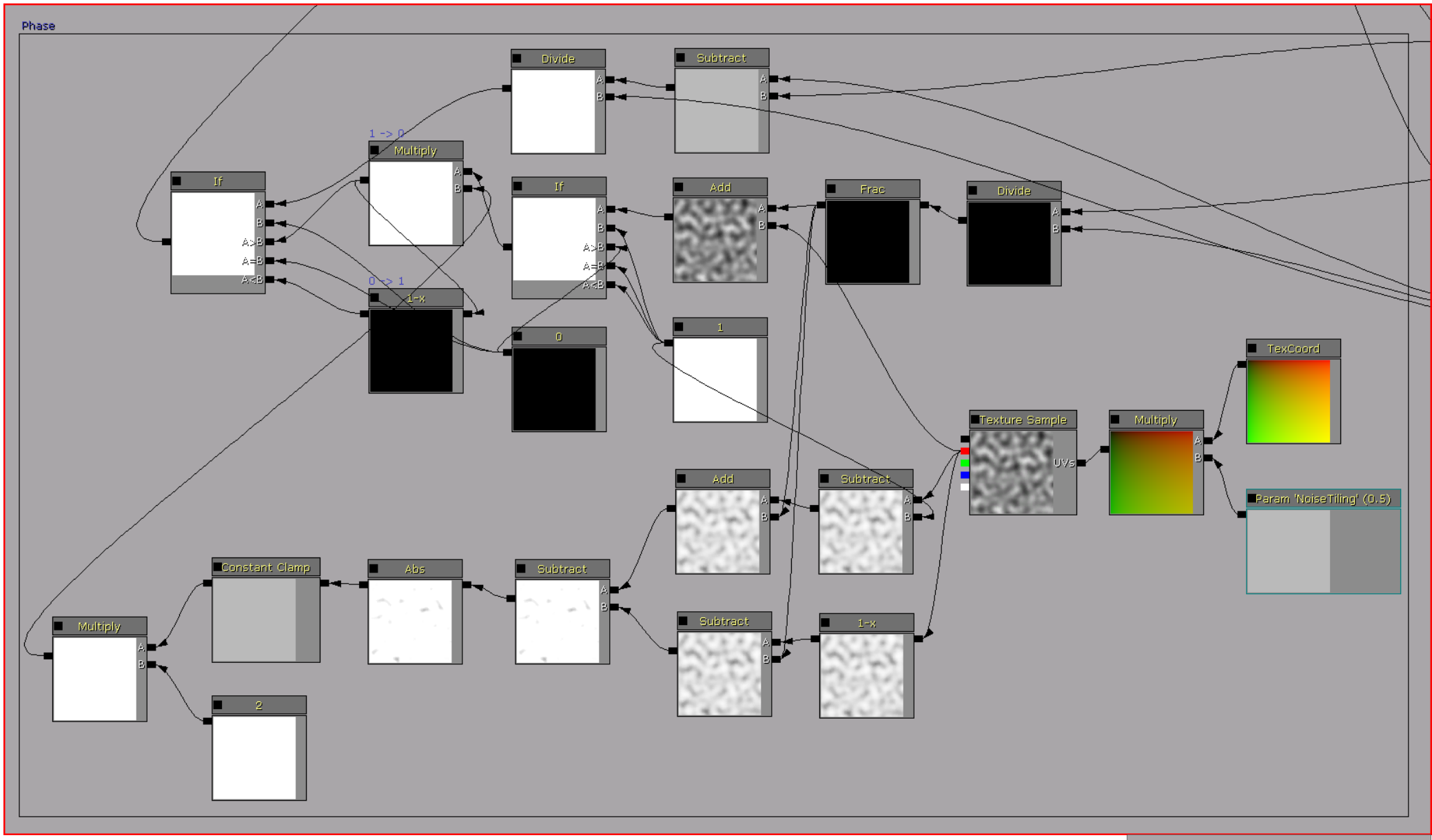


So,

Time = 0,	Phase = 1,	2nd map is fully visible,
0 < Time < 0.5,	Phase – growing black spots,	1st map is partially visible,
Time = 0.5,	Phase = 0,	1st map is fully visible,
0.5 < Time < 1,	Phase – growing white spots,	2nd d map is partially visible,
Time = 1,	Phase = 1,	2nd map is fully visible.

Material layout:

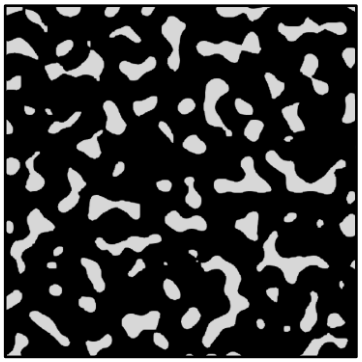



Phase
Divide
Subtract
1 -> 0
Multiply
If
If
Add
Frac
Divide
0 -> 1
1-x
0
1
TexCoord
Texture Sample
Multiply
UVs
Add
Subtract
Param 'NoiseTiling' (0.5)
Constant Clamp
Abs
Subtract
Multiply
Subtract
1-x
2

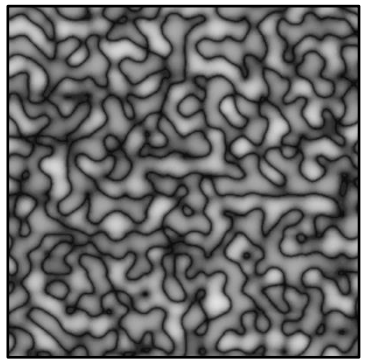
`Frac(Time/0.5)`, to get 2-times faster saw-tooth function.

`(Noise - 1)`, to get noise < 0 (-1...0), and inverted noise > 0 (0...1).
`((Noise < 0) + Frac(Time/0.5))`, to raise it with time to 0...1.
`((Noise > 0) - Frac(Time/0.5))`, to lower it with time to (-1...0).

Hard-edged



Soft-edged



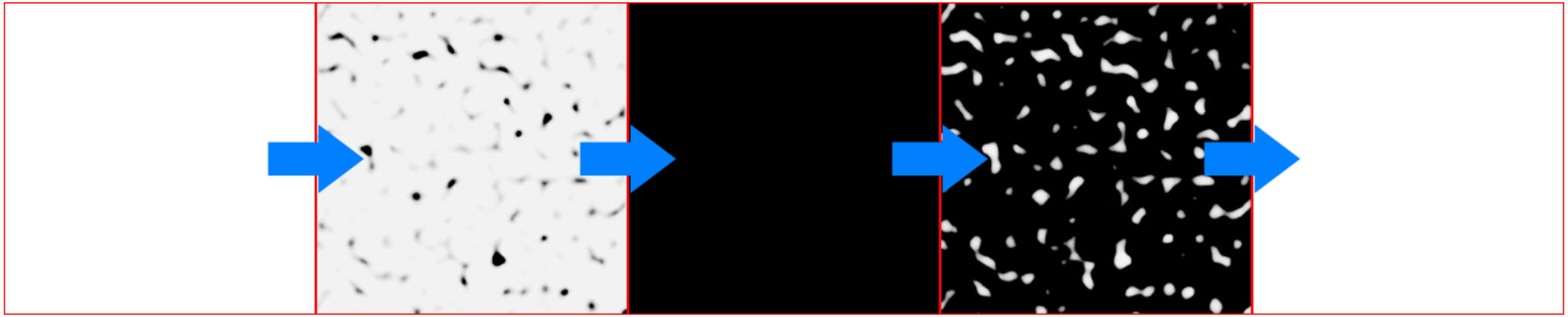
`If(Frac(Time/0.5) + (Noise > 1))`, to get hard-edged mask.

`Abs((Noise < 0) - (Noise > 0))`, to get soft-edged mask.

Then `Clamp(0, 0.5)` and multiply on 2, to modify soft-edged mask.

And multiply hard-edged mask on soft-edged mask to get final result.

Time = 0 | 0 < Time < 0.5 | Time = 0.5 | 0.5 < Time < 1 | Time = 1

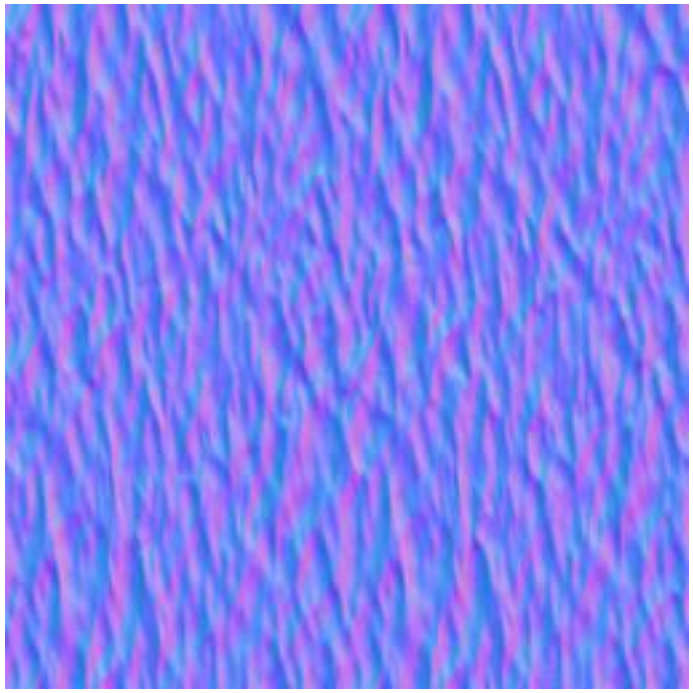


When you will see result of this operation, you may think that it more noticeable than previous surface pulsing.
I think so too.
Surface's pulsing mostly produced by strong normals and specular lightning, so keep that in mind.

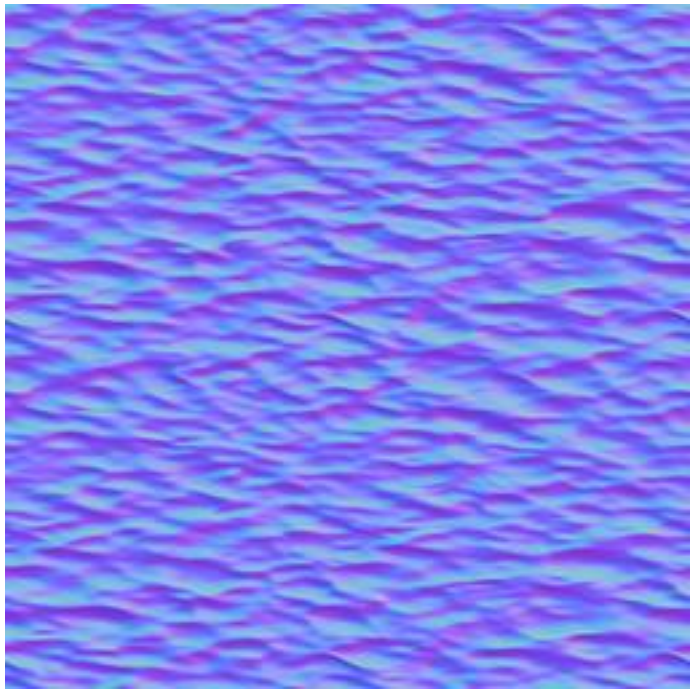
Texture direction

Your flowing textures should correspond to flow direction.
Here example with special U and V directed normal maps.

Normal U

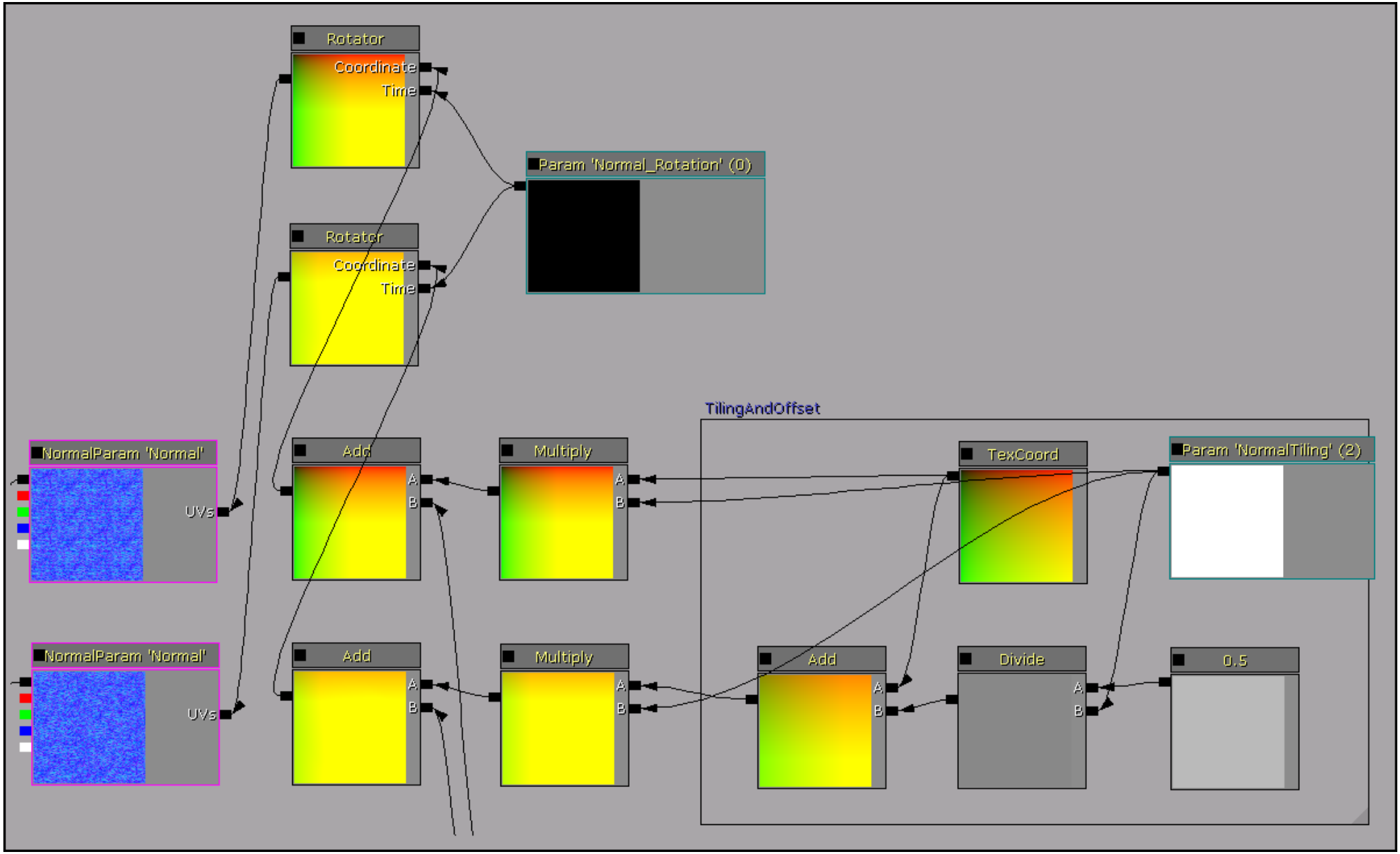


Normal V



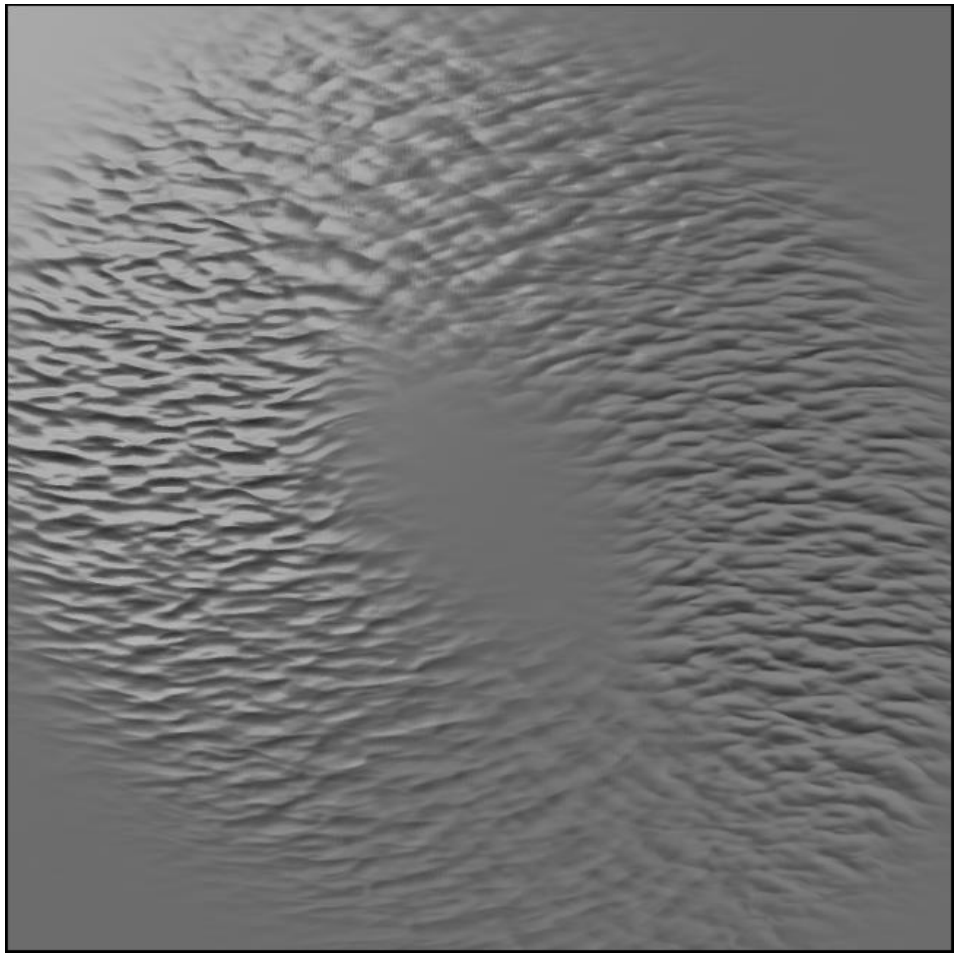
This textures works fine with upwellings and vortices, but you always have ability to experiment with others.

Also, you can rotate textures to main flow direction.

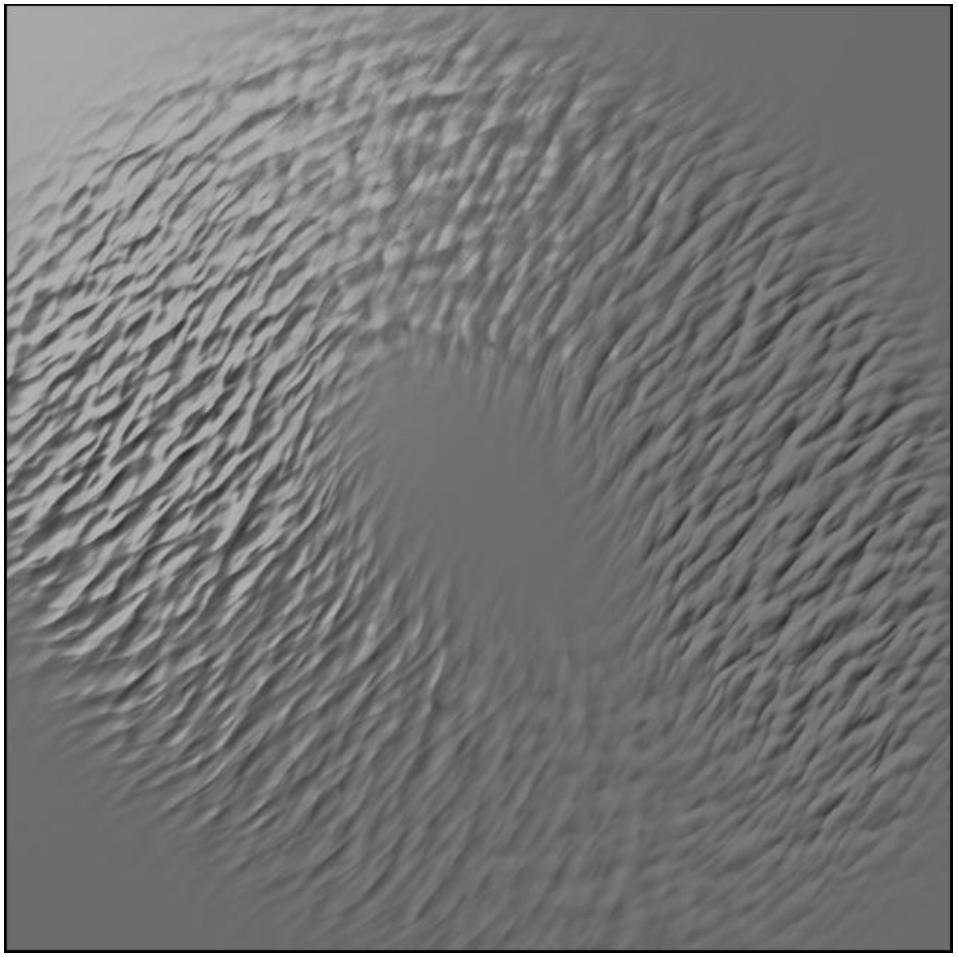


It will provide better visual results.

Rotation = 0



Rotation = 0.785398 (45°)

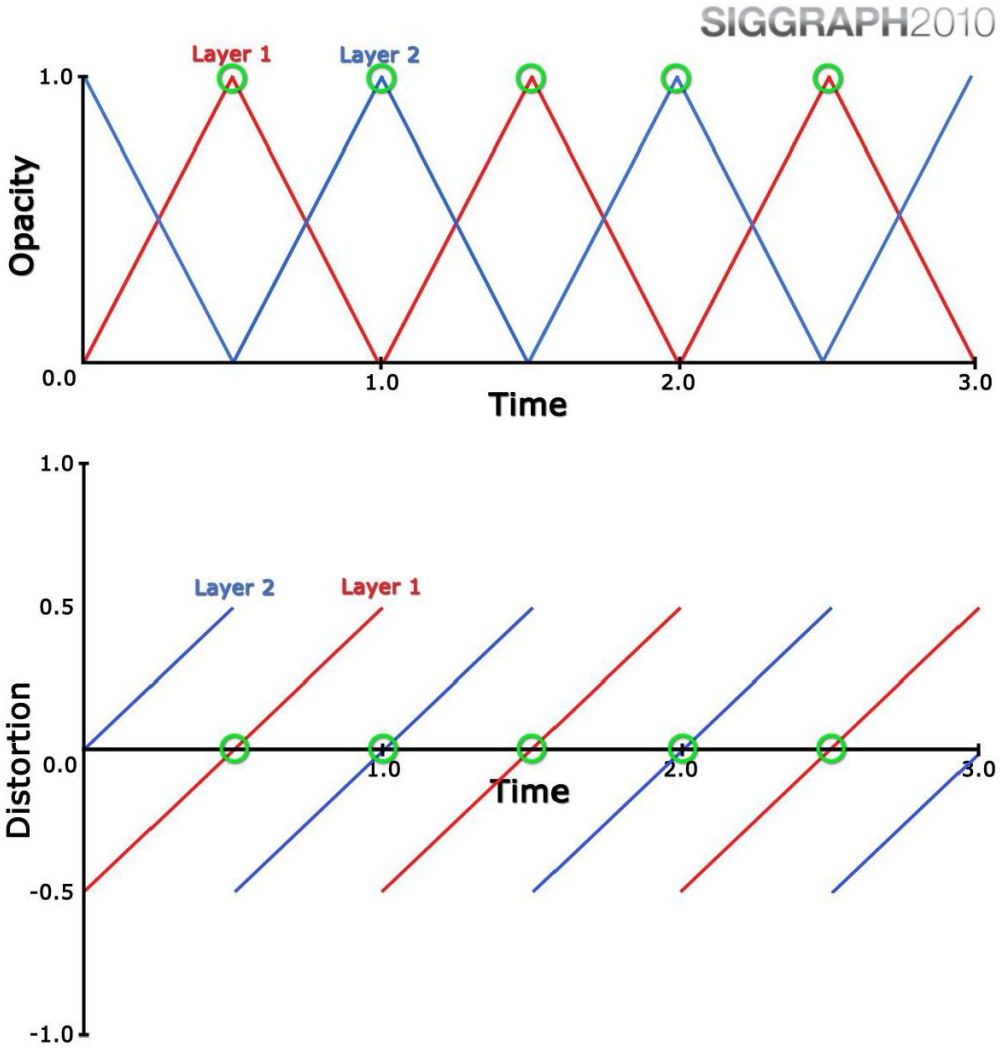


Flowing color maps

Again, page from [Water Flow in Portal 2]

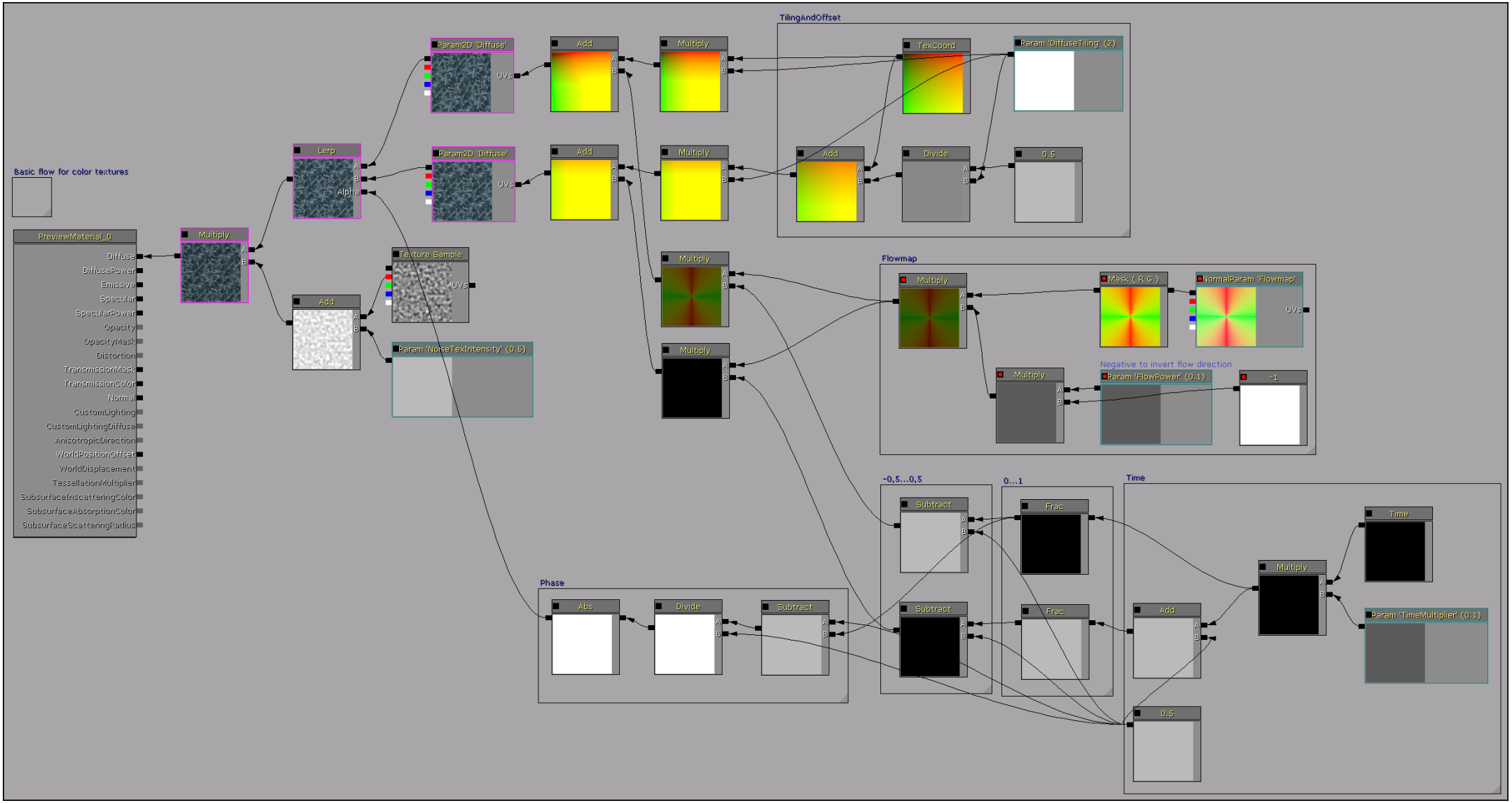
Flowing Debris

- Flowing colors works better by offsetting the interval from –`fraction` to `+fraction` so the peak of the interval is at zero (the at-rest position)



Basic flow for color maps

Material layout:

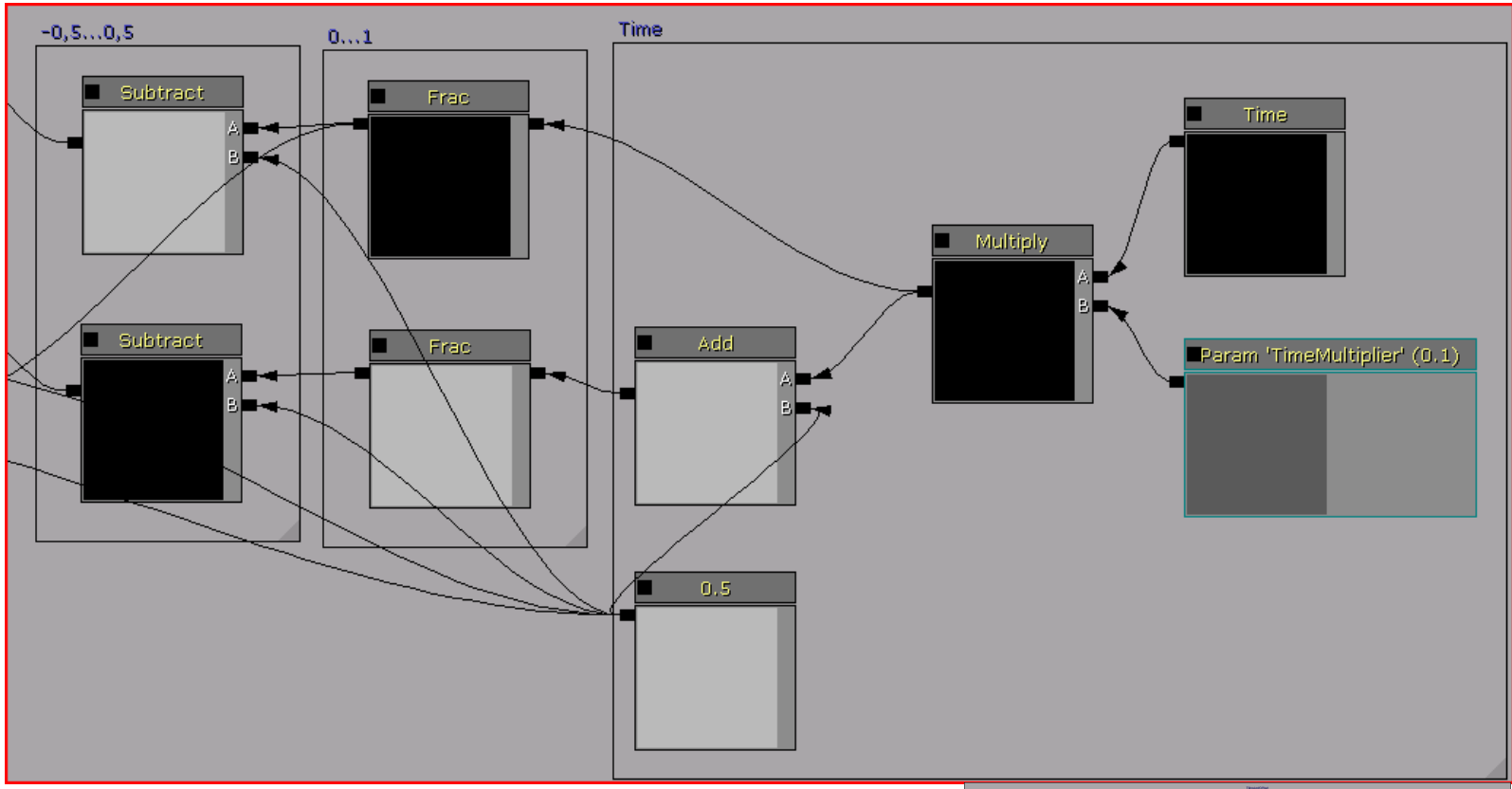


Almost the same as for normal maps.

For color maps, distortion range should be -0.5...0.5.

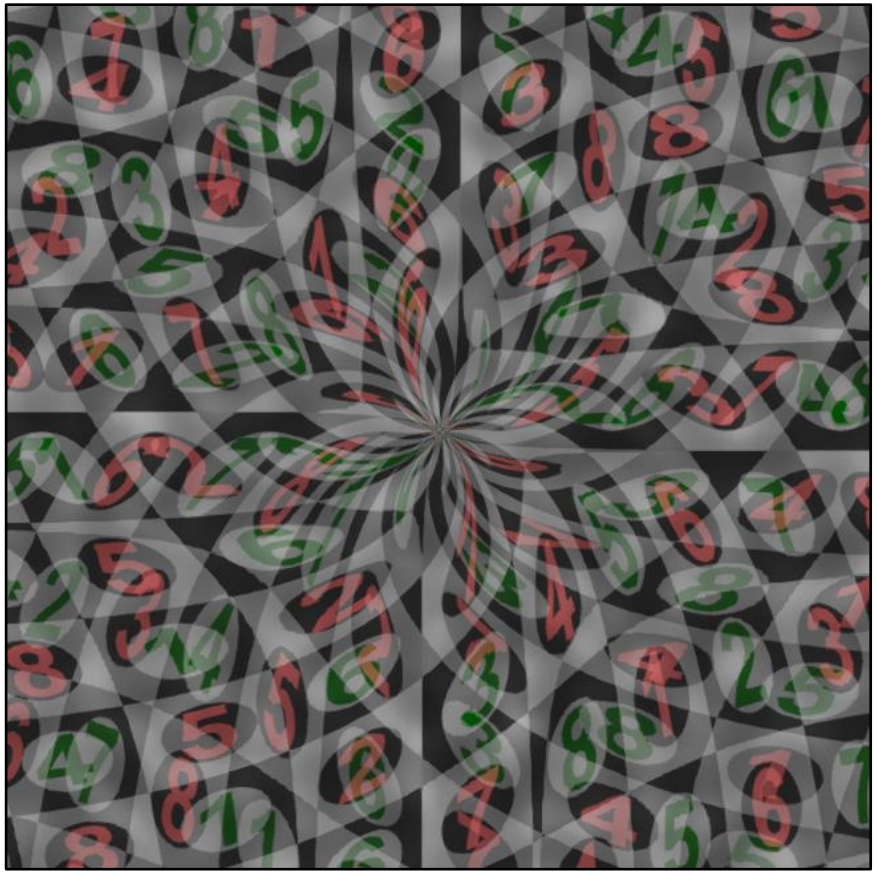
So,

```
Frac(Time*Mult) - 0.5
Frac(Time*Mult + 0.5) - 0.5
```

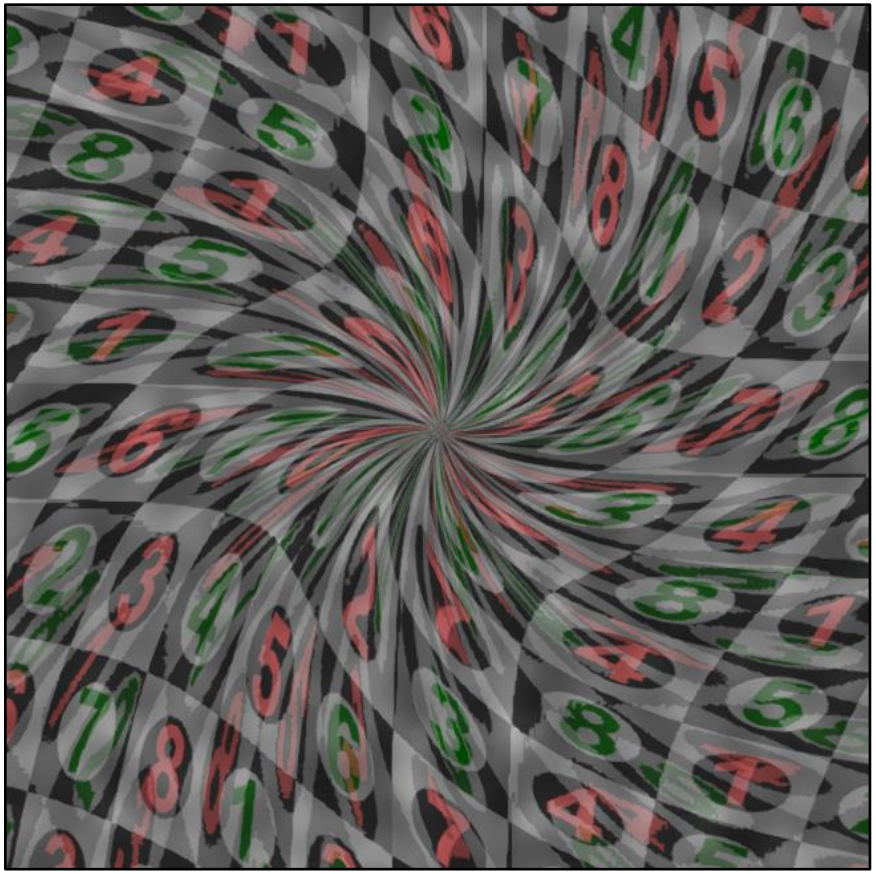


But, sometimes it's not needed, depends on desired effect.

-0.5...0.5

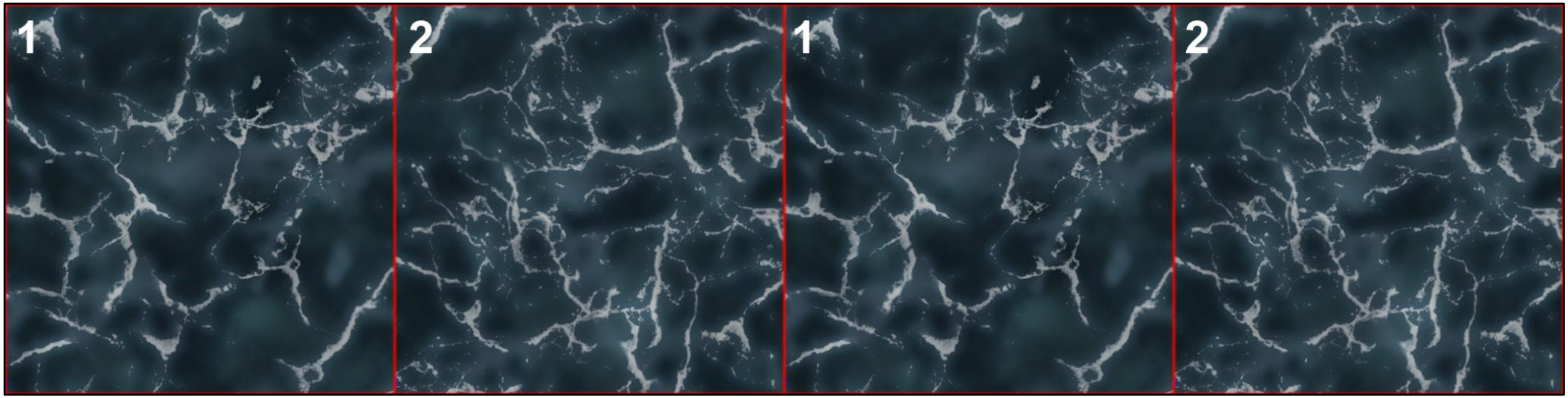


0...1

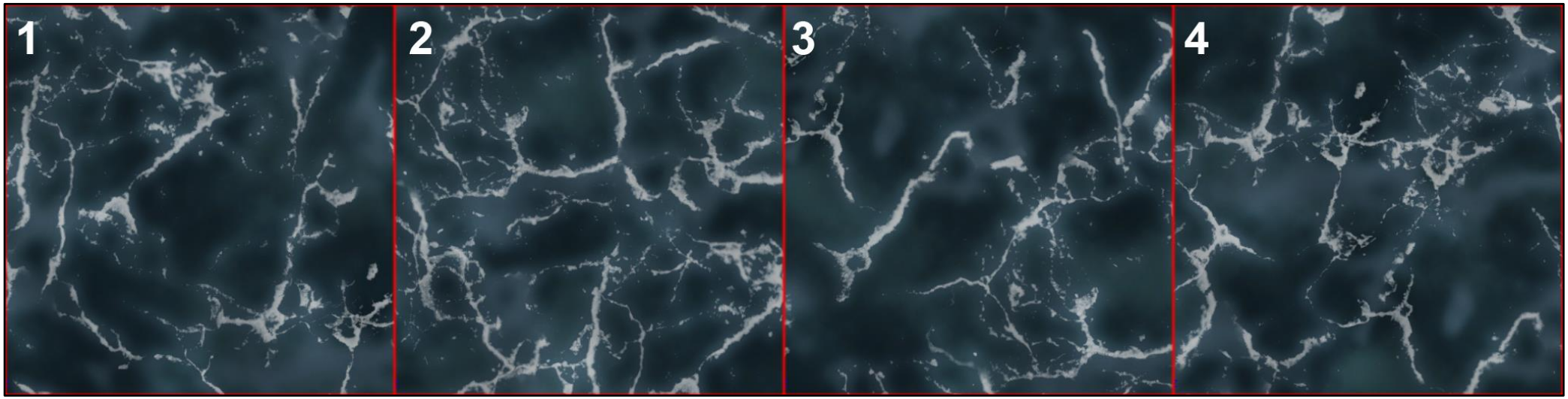


Basic flow for color maps with instantaneous UV offset.

Using previous setup, you can notice texture's repetition from phase to phase.

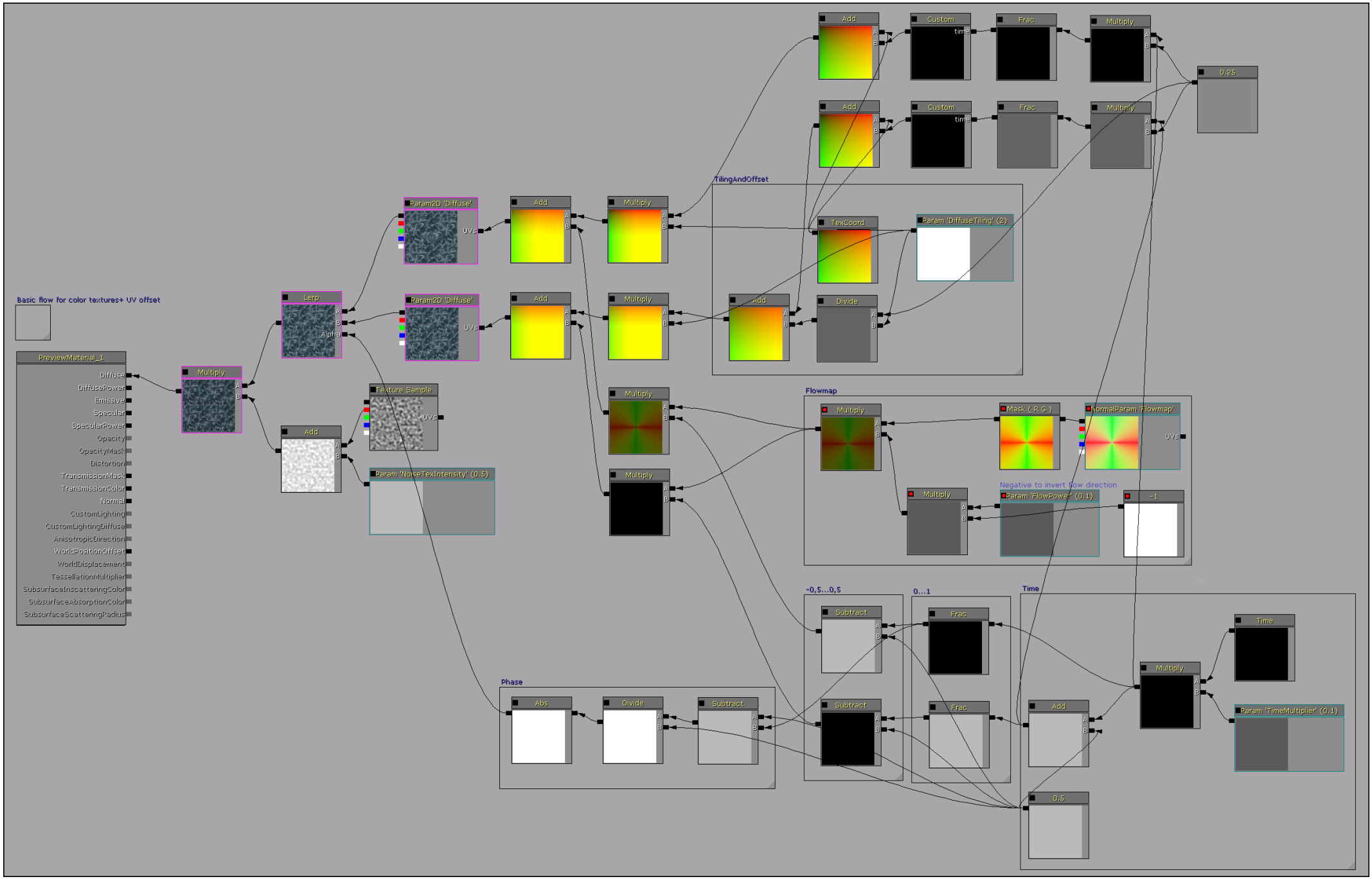


Here you can use one trick - when phase changes, and texture is fully hidden, it UVs instantaneously offsets on 0.25, and now, repetition happens only after phase*4.



But not 4 phases, only 2, because your 2nd texture shifted on 0.5 from 1st.
It will be 4 if you use different textures.

Material layout:

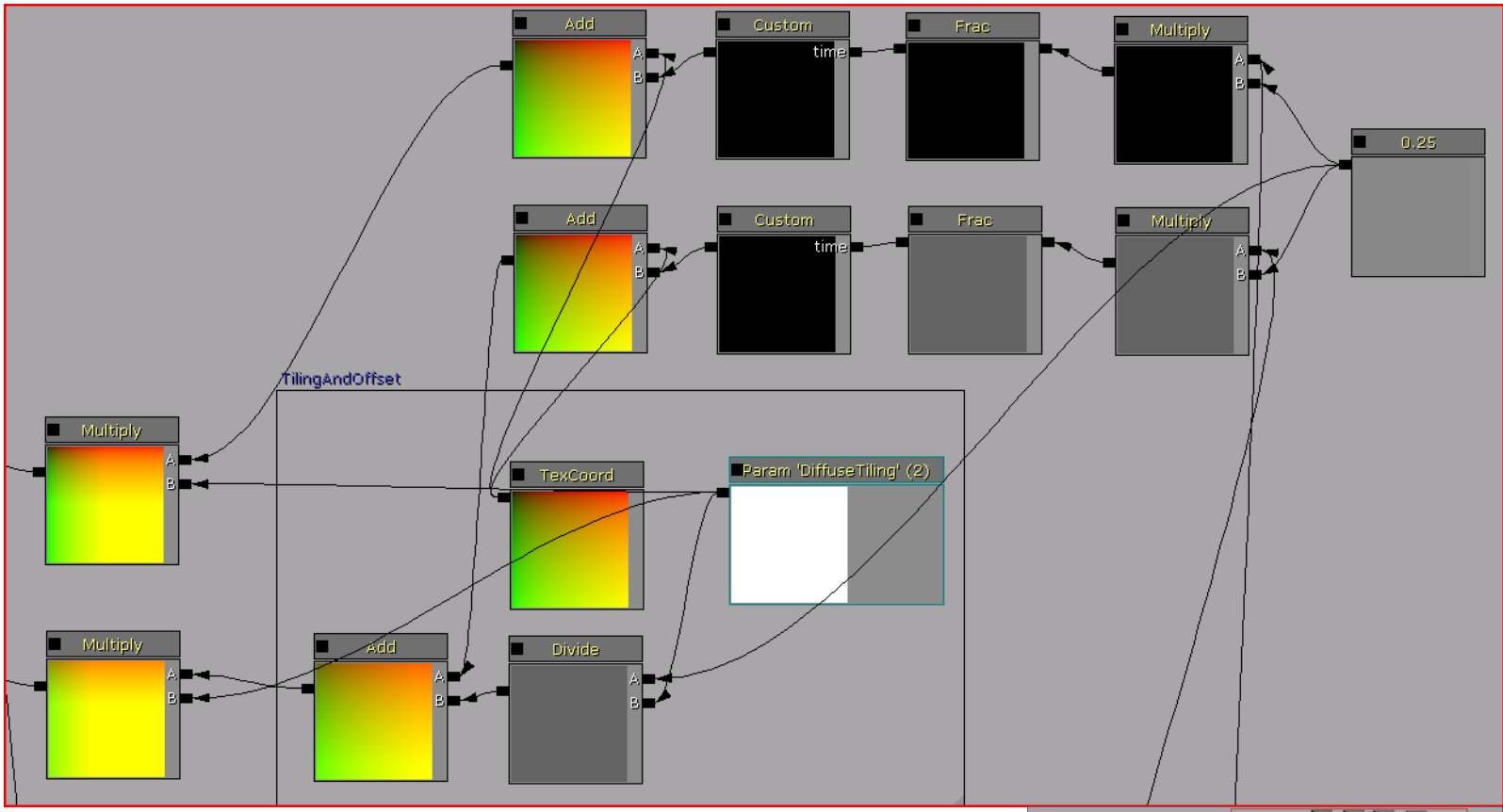


Here used custom HLSL code.

```
float output;

if(time>=0 && time<0.25)
{
        output = 0;
}
else if(time>=0.25 && time<0.5)
    {
        output = 0.25;
    }
    else if(time>=0.5 && time<0.75)
        {
            output = 0.5;
        }
        else if(time>=0.75 && time<1)
            {
                output = 0.75;
            }

return output;
```

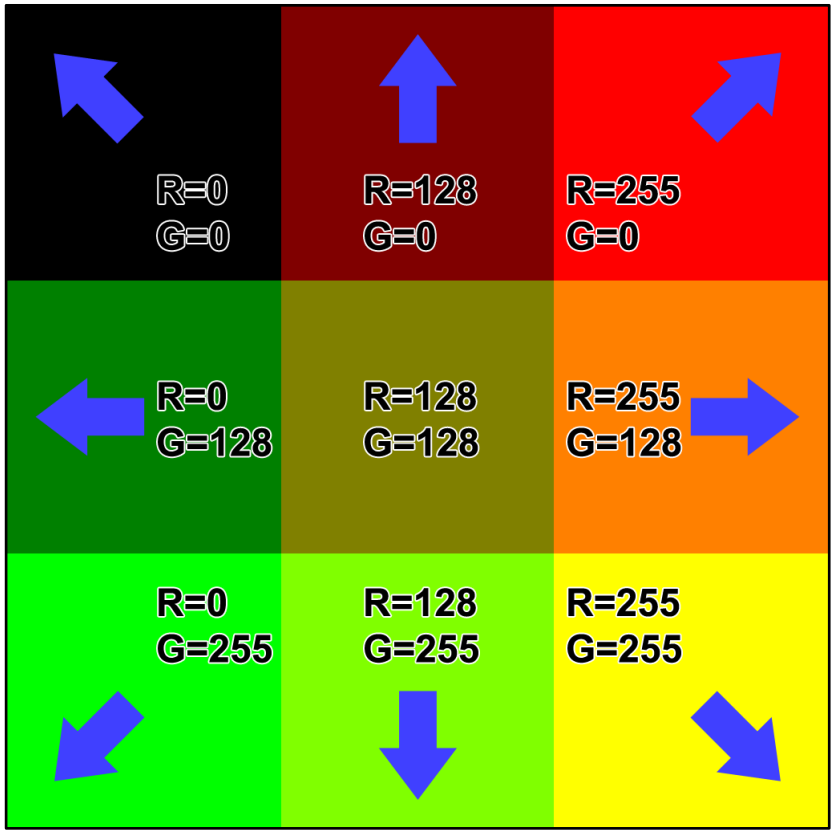


Of course, you can distort color maps by normals, add reflections with refractions, some depth-biased opacity with depth fog, and special lightning effects.
This tutorial provides only basic instructions.

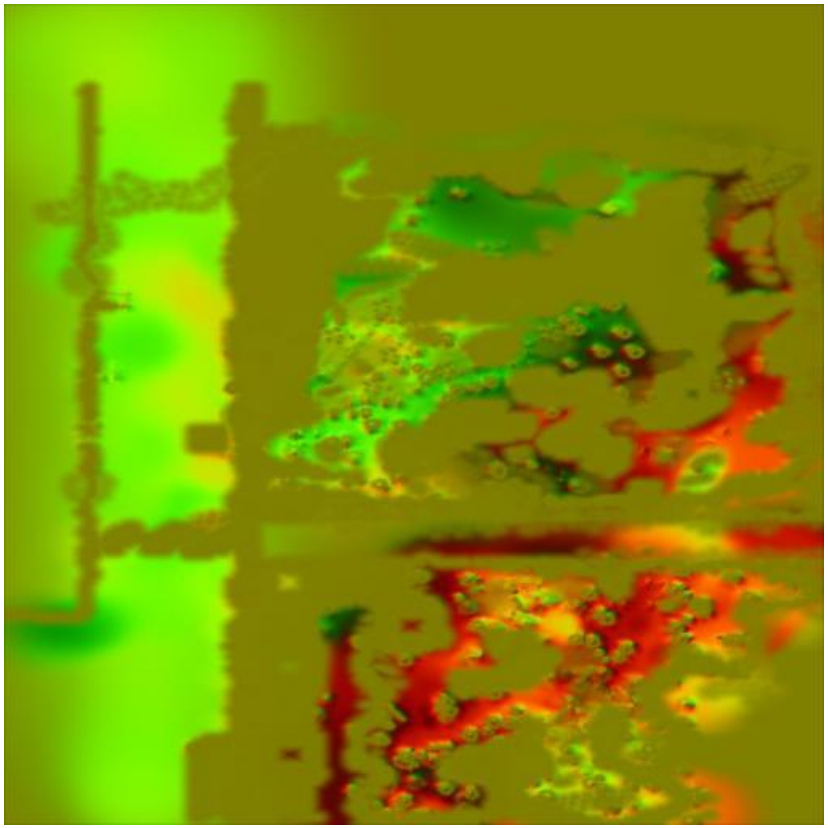
Flow map creation

Now let's see how you can create flow maps
To do this, you need some XY vector field, represented as texture and (!) unpacked from -1 to 1, like normal maps.

This scheme explains how flow vectors looks in colors.



Can you draw it?

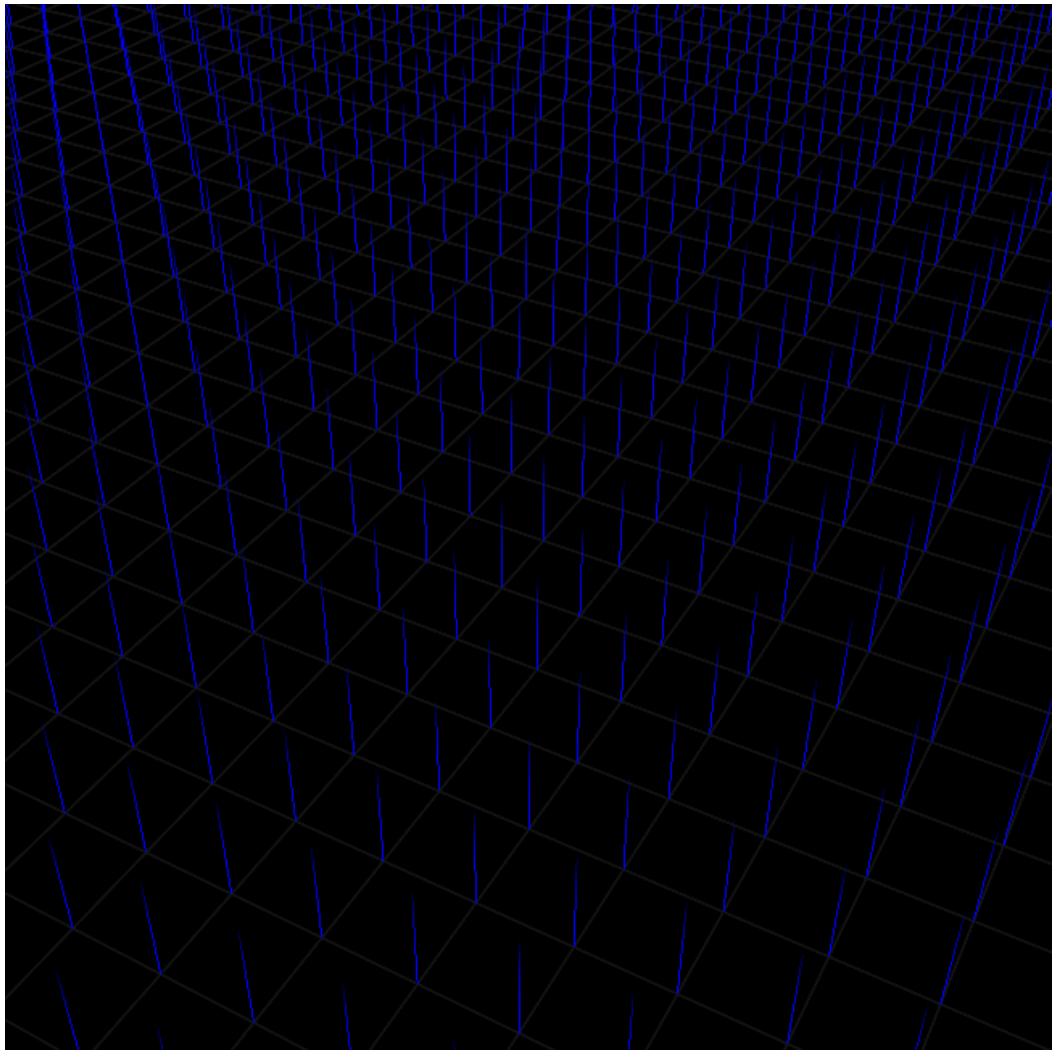


For this tutorial used **Houdini** by Side Effects Software.
Of course, you need some basic knowledge of **Houdini** to understand next chapter.
But I tried to explain every step.

Houdini basics

Creating vector field

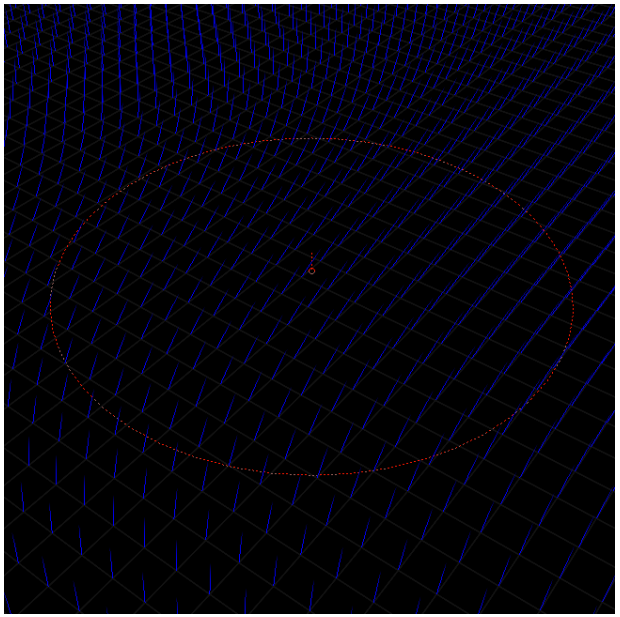
Using **Houdini,** you can create simple tessellated plane (**Grid**), with it's points normals as vectors.



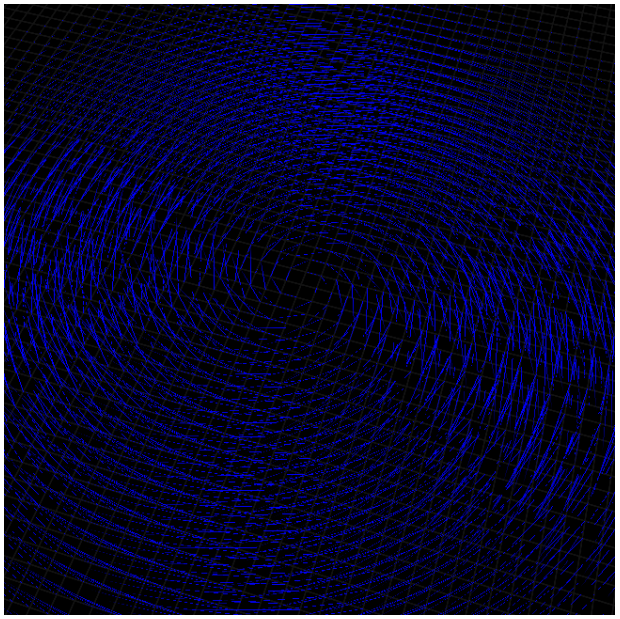
By default, **Houdini** use Y-up coordinate system.
So, if normal is Y-up, it means no flow. But if normal is tilted in some direction, it's X and Z values are different from 0.

Houdini provides many ways to manipulating normals.

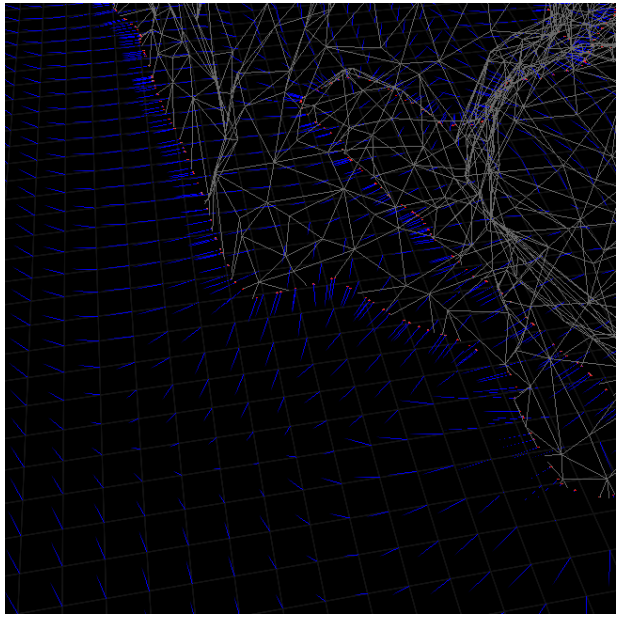
1st – **Comb** tool. It works like brush, which tilts normals.



2nd – calculations with deformers.

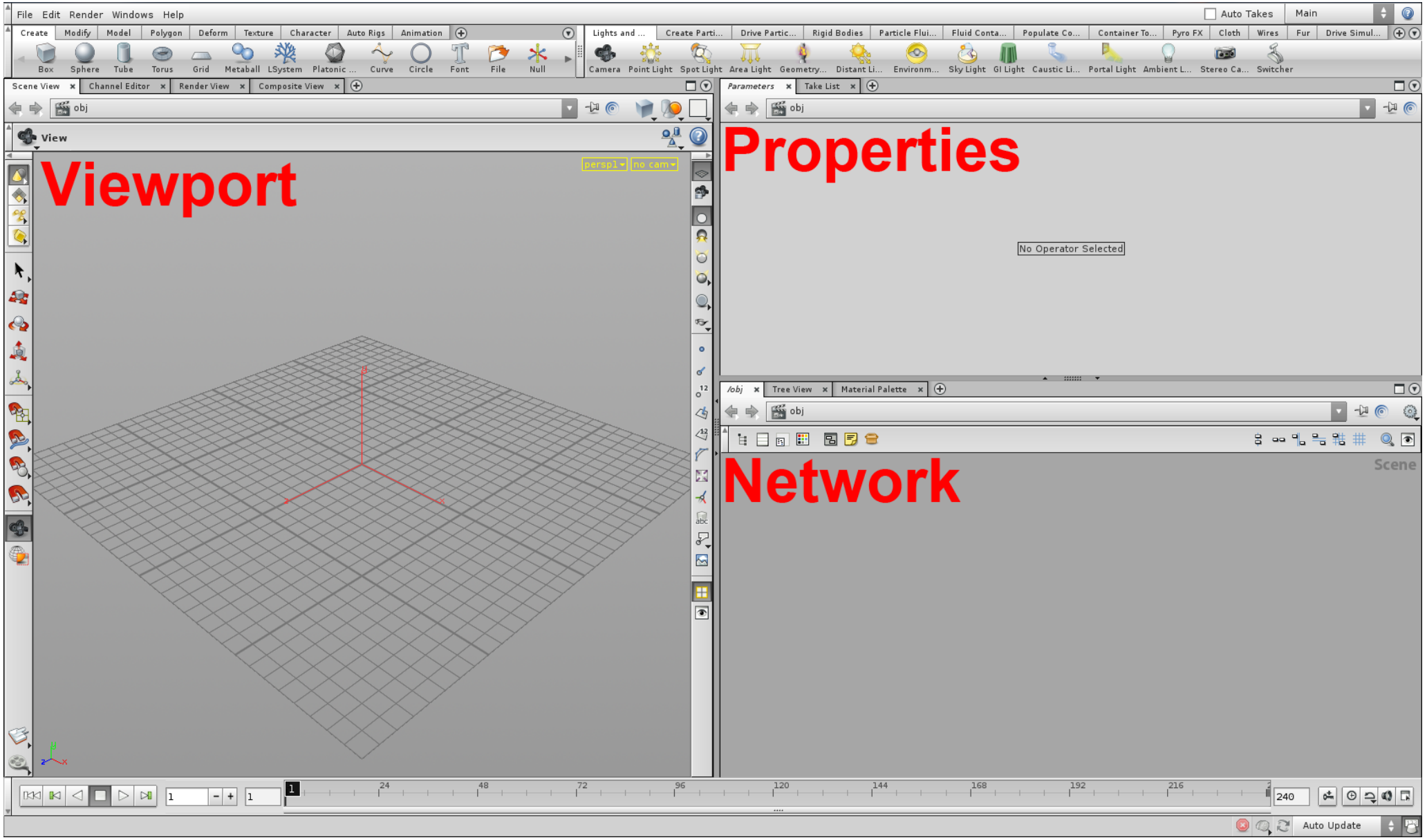


3rd – transferring attributes from one object to another.



Combing normals

Open **Houdini**.

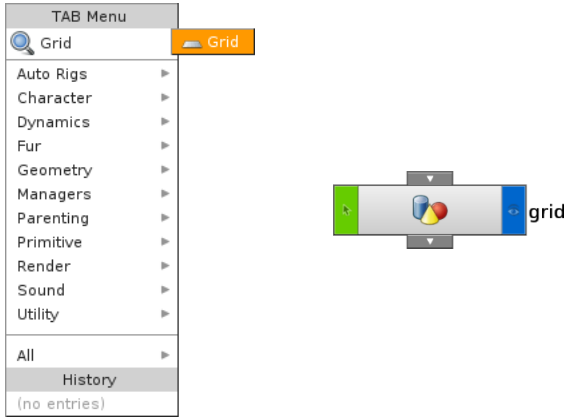


Click on “Shading options”  on top side of **Viewport**, and select “Flat Wire Shaded”.

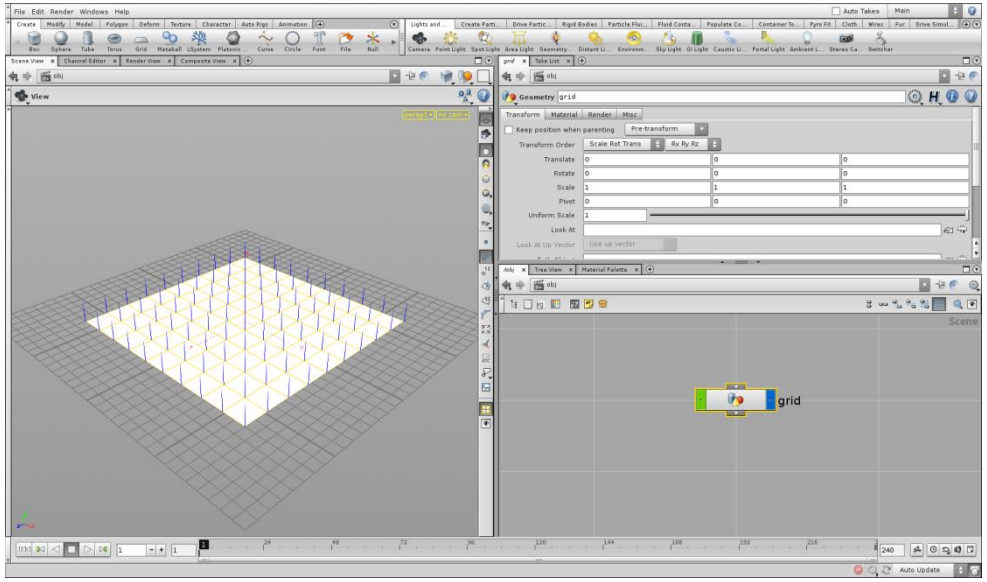
Click on “Display point normals”  on right side of **Viewport**, to display point normals.

Click on “Show/hide grid and enable/disable snapping”  on top side of **Network** view, to activate nodes snapping grid.

Then, hover mouse on **Network** view, press TAB and type “Grid”. When **Grid** selected, press Enter and place grid object.

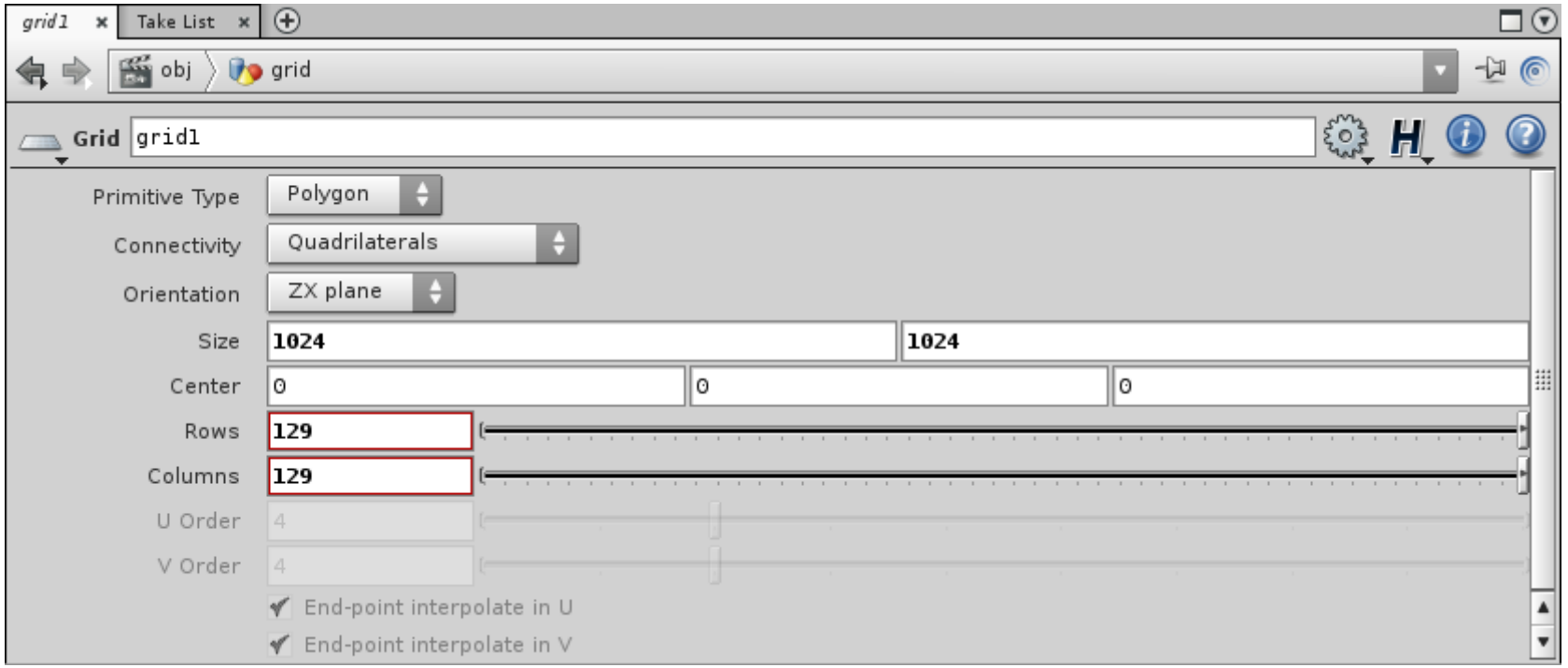


Now you can see some grid without any transforms, and with it’s points normals.

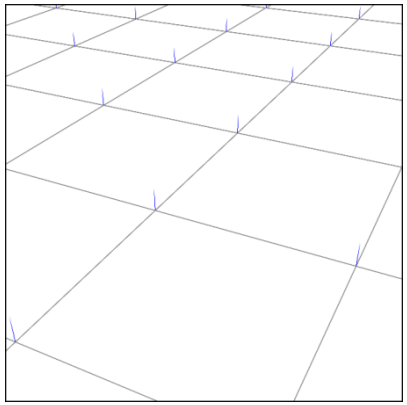


Double click on it in **Network** view to open. Select internal **Grid** object, and adjust it's Size, Rows and Columns parameters.

Orientation = ZX plane.
Size = 1024x1024.
Rows/Columns = 129x129.



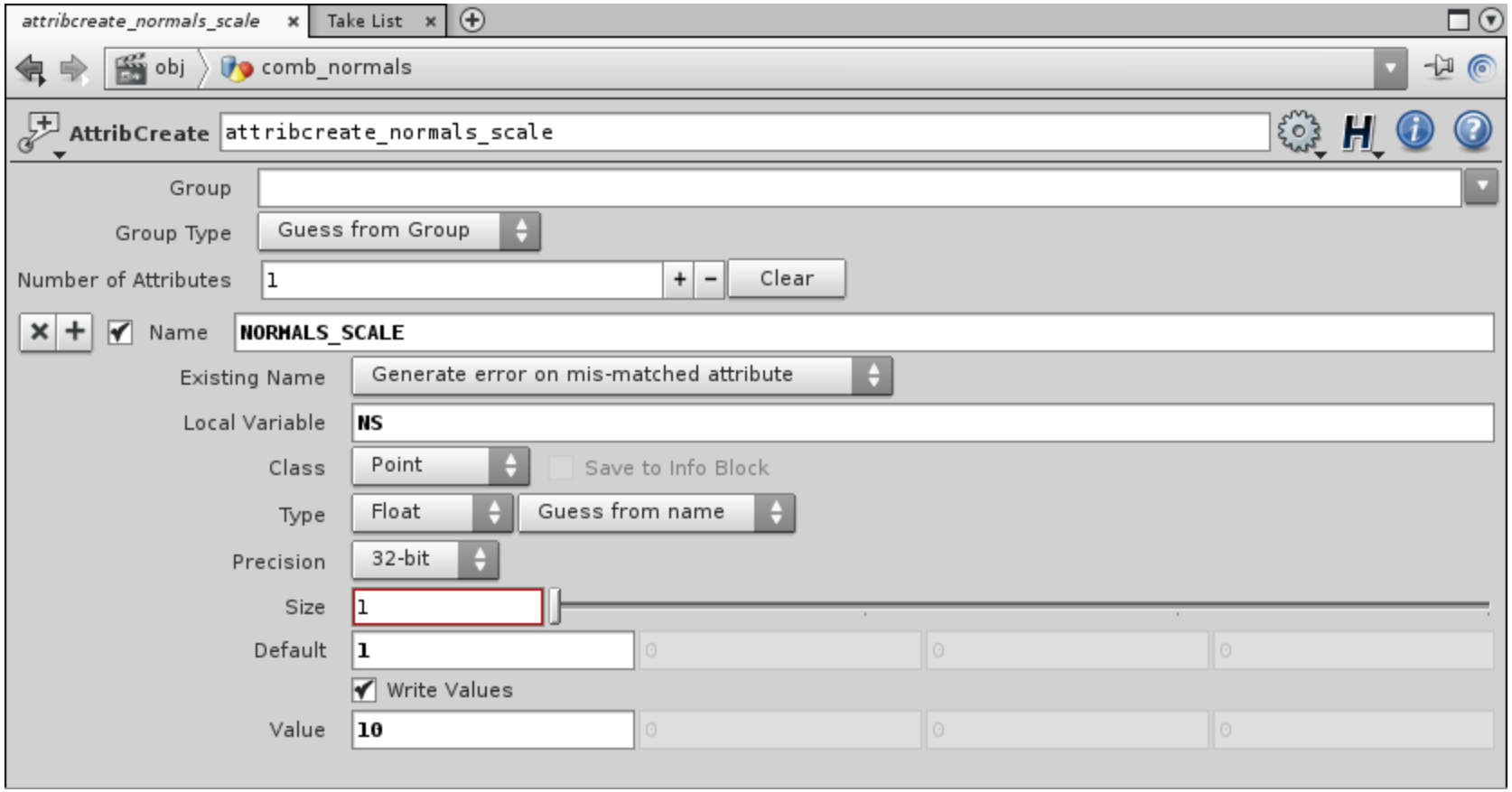
Now, your grid is very large, but it's normals are very small. Let's fix that.



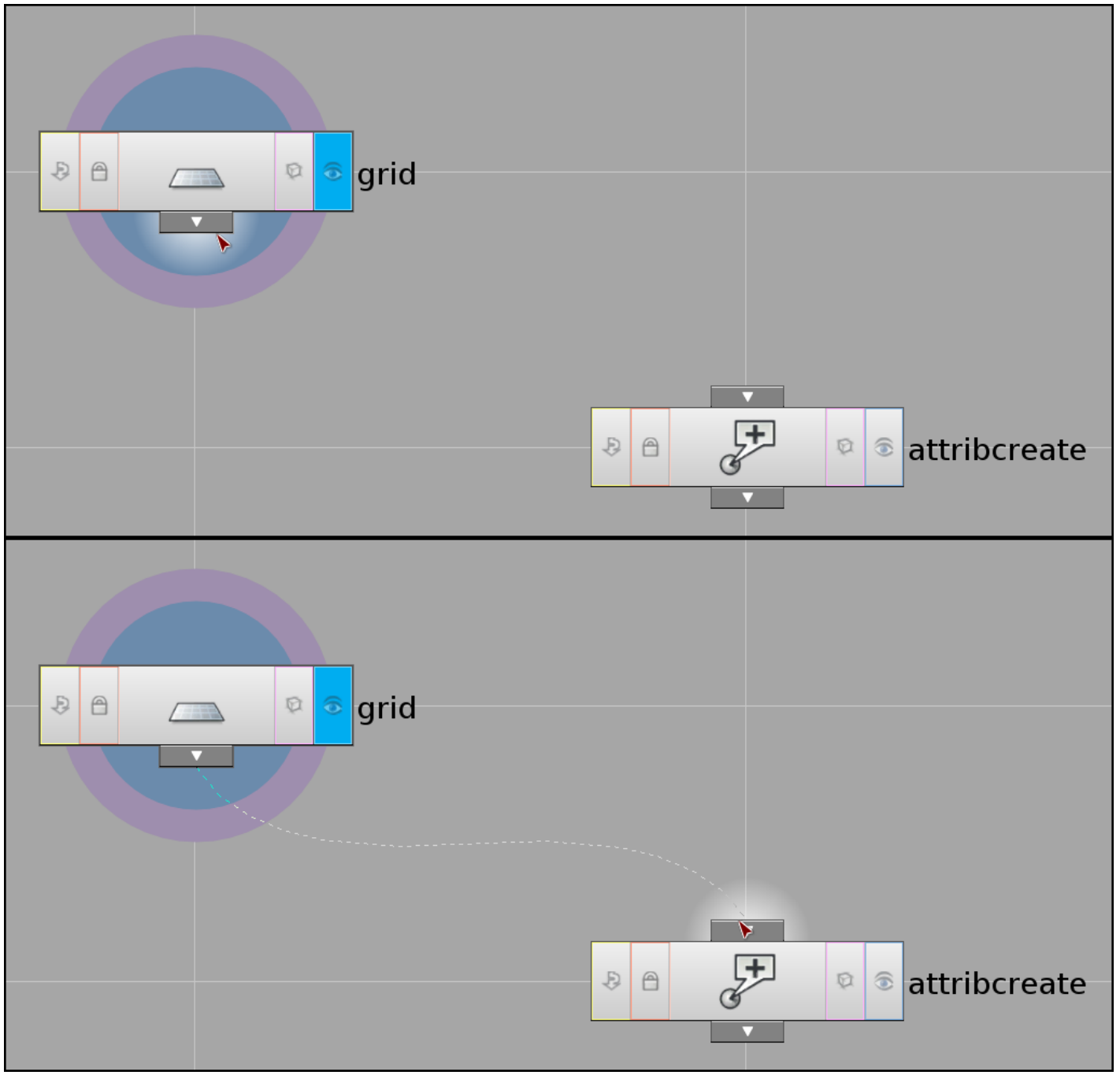
Again, **Network** view -> TAB -> **AttribCreate**

This node can be used for creating new attribute for your network.

Name = NORMALS_SCALE.
Local name = NS.
Class = Point.
Type = Float.
Value = 10.



Click on **Grid**'s output, and connect with **AttribCreate** input.

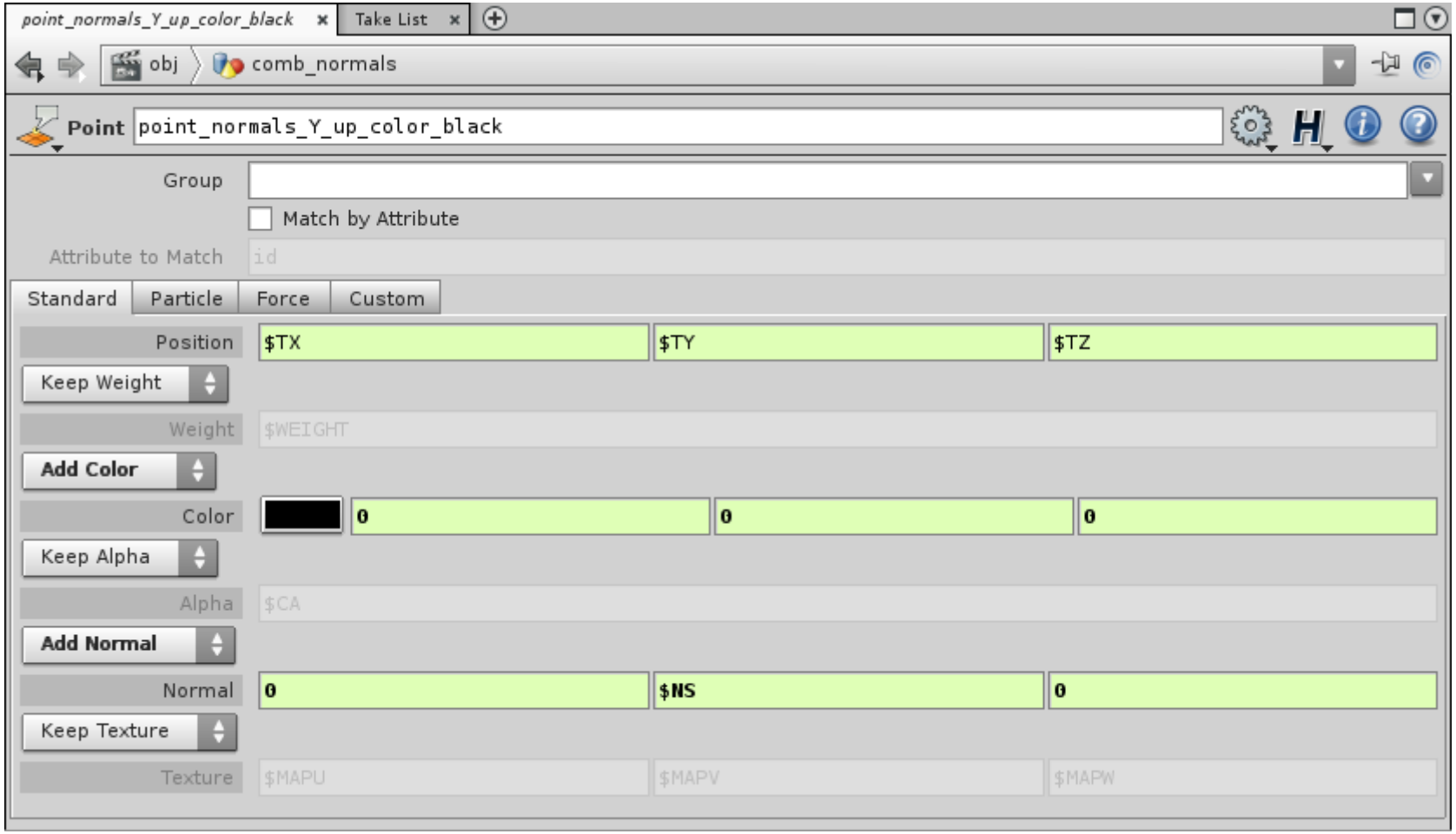


How you can use this new attribute for scaling normals?

TAB -> **Point**. 

This node operates various point attributes, as Position, Color, Normal, etc.
In this case you need 2 things – set points color black, and scale your normals to NORMALS_SCALE (NS) value, to make it more noticeable.

Color -> Add Color = 0, 0, 0.
Normal -> Add Normal = 0, $NS, 0.



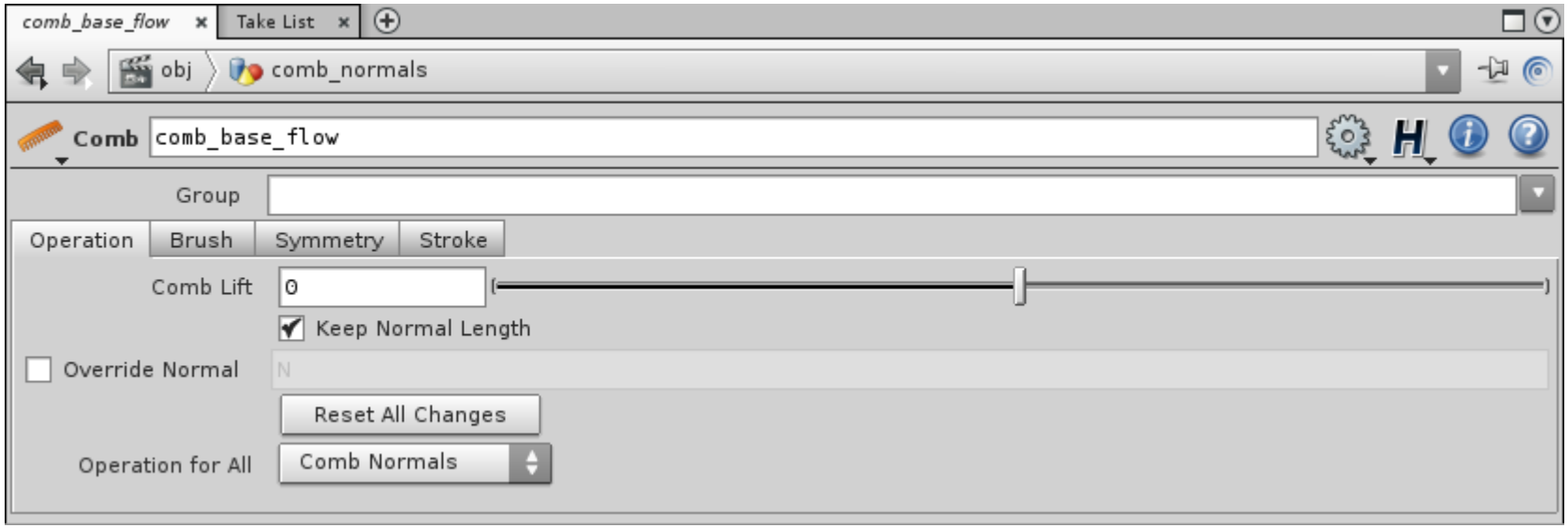
Again, connect **AttribCreate** output with **Point**'s 1st (left) input.

Click on “display” icon on **Point** node, to view result of current operation.

Now, you must see black grid with big noticeable normals. Like on 1st picture in this chapter.

TAB -> **Comb**. 

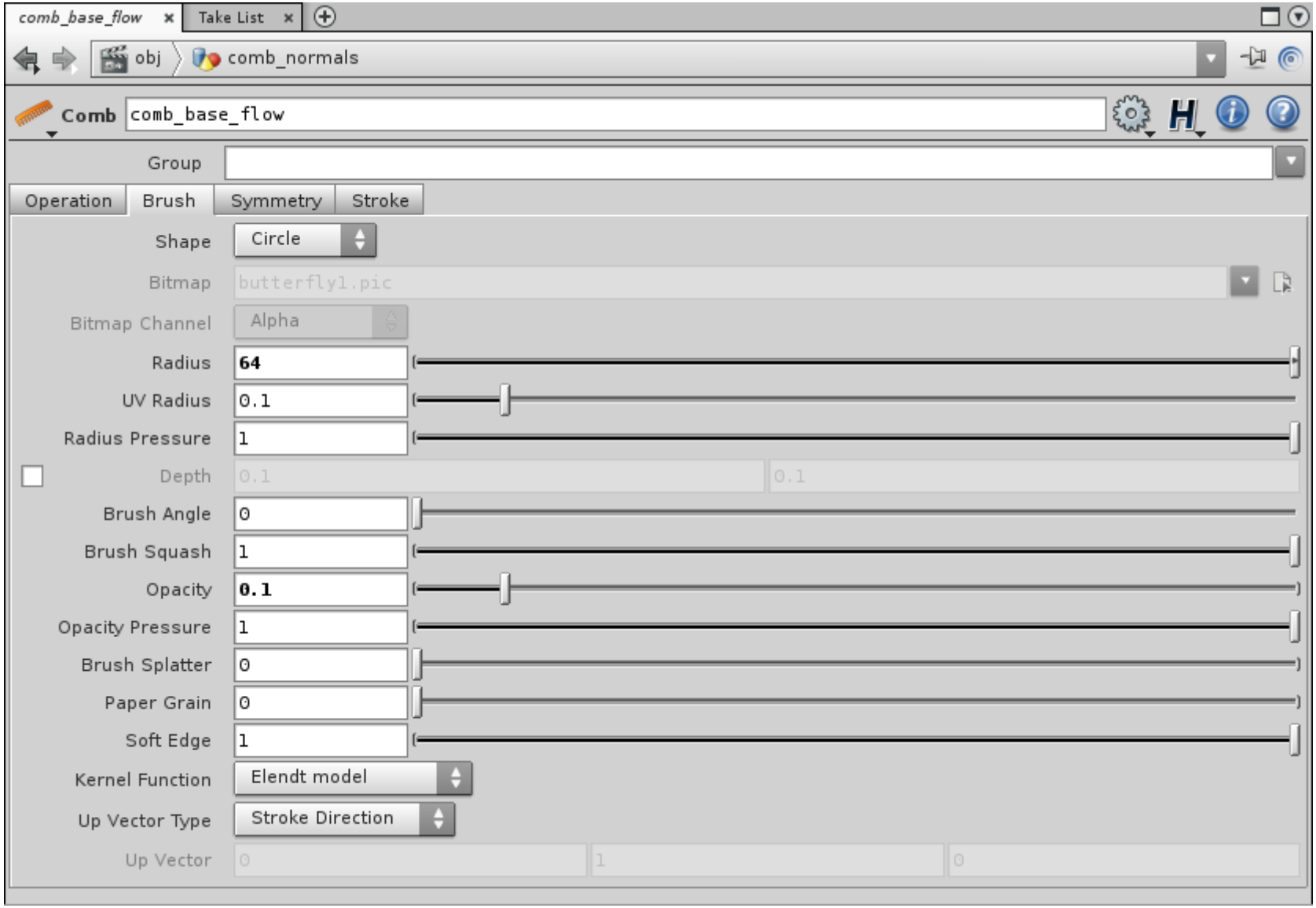
Operation = Comb Normals.



“Reset All Changes” button erase panted comb and return normals to previous state.

Select **Comb** node, hover **Viewport** and press Enter.
Now your cursor changed to circle of brush.

Brush – your custom settings.



I hope you will quickly learn this tool.

Comb something with LMB.

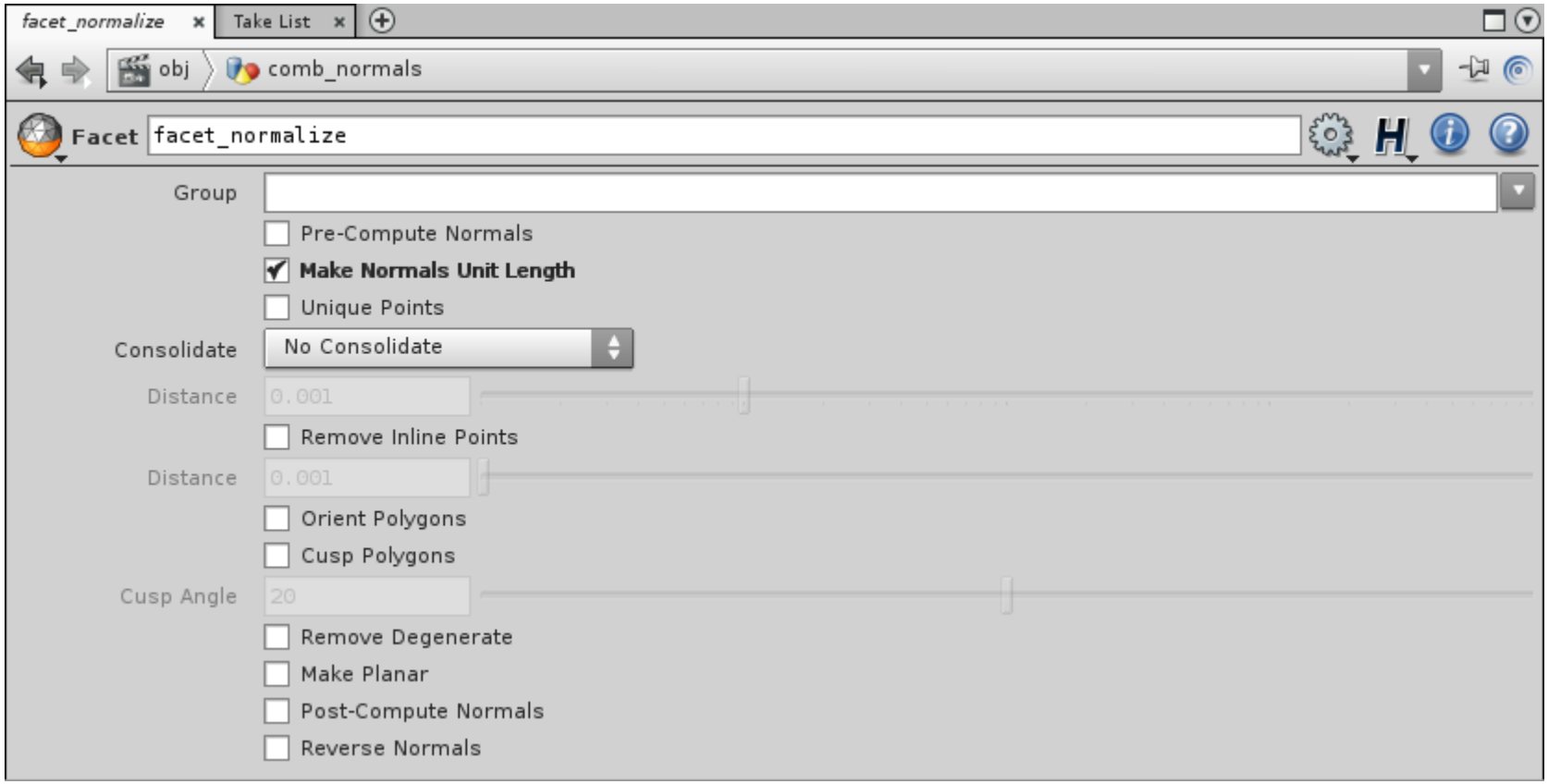
Hover **Viewport** and press Esc to finish operation.

Now, you need to get your painted flow as a texture.

TAB -> **Facet**. 

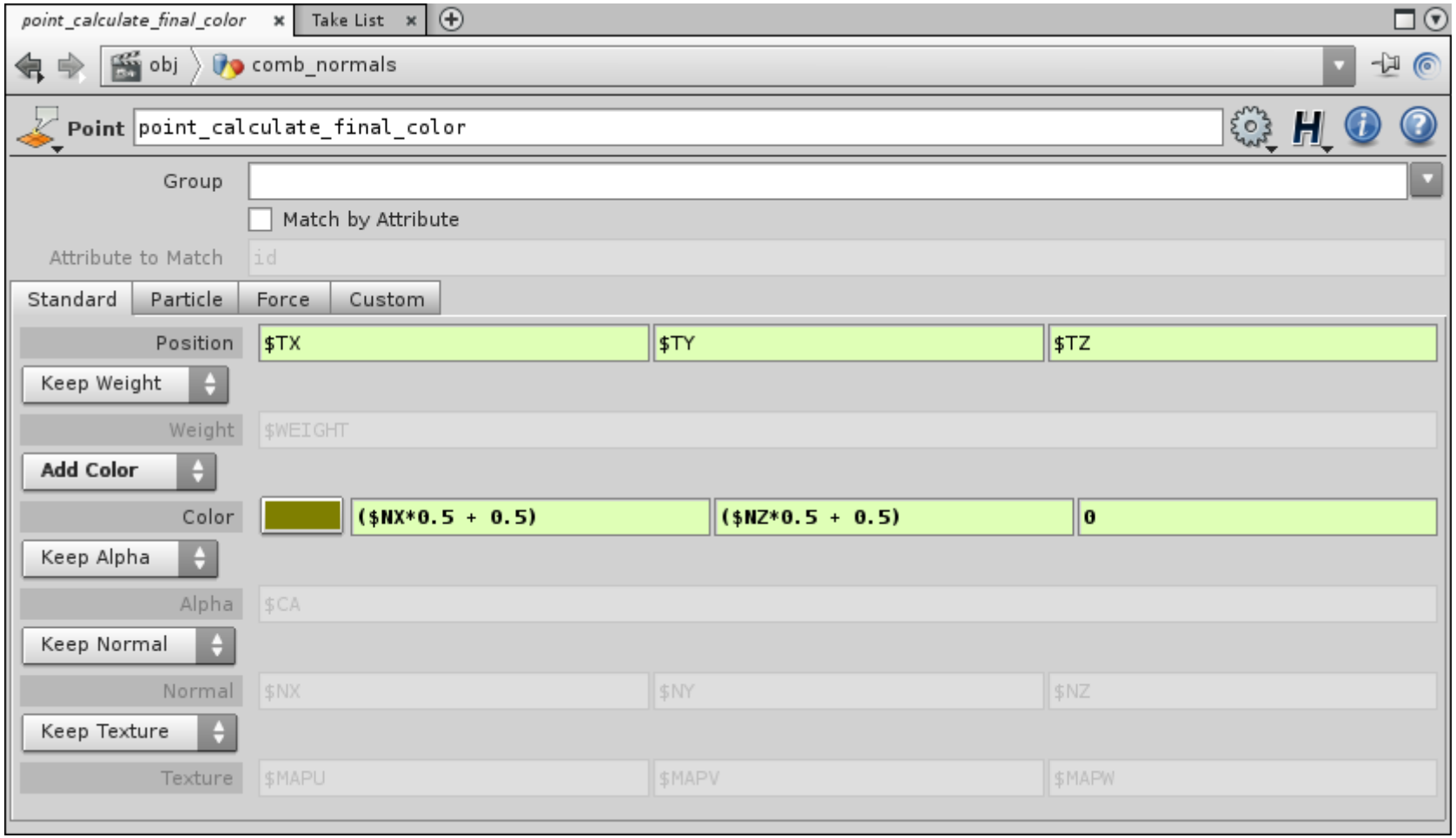
This tool also works with normals. Using it you can Pre-Compute, Reverse, or normalize your normals.

So, check “Make Normals Unit Length”. This will set normals length to 1.

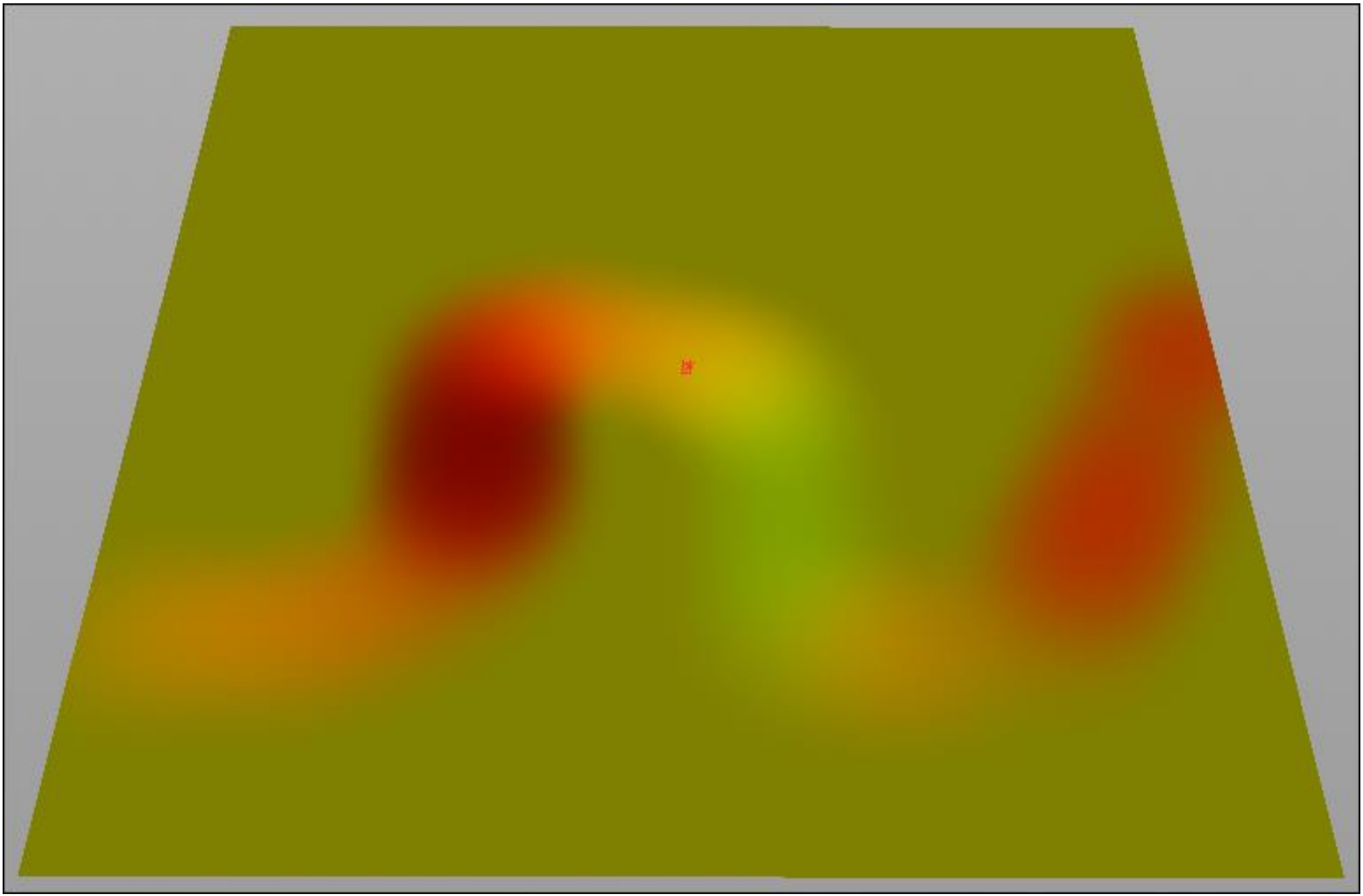


In this case, normals length = NS.
You will get identical result if use **Point** -> Add Normal = $NX/$NS,
$NY/$NS,
$NZ/$NS.

Finally,
TAB -> Point
Color -> Add Color = $NX*0.5 + 0.5,
$NZ*0.5 + 0.5,
0.

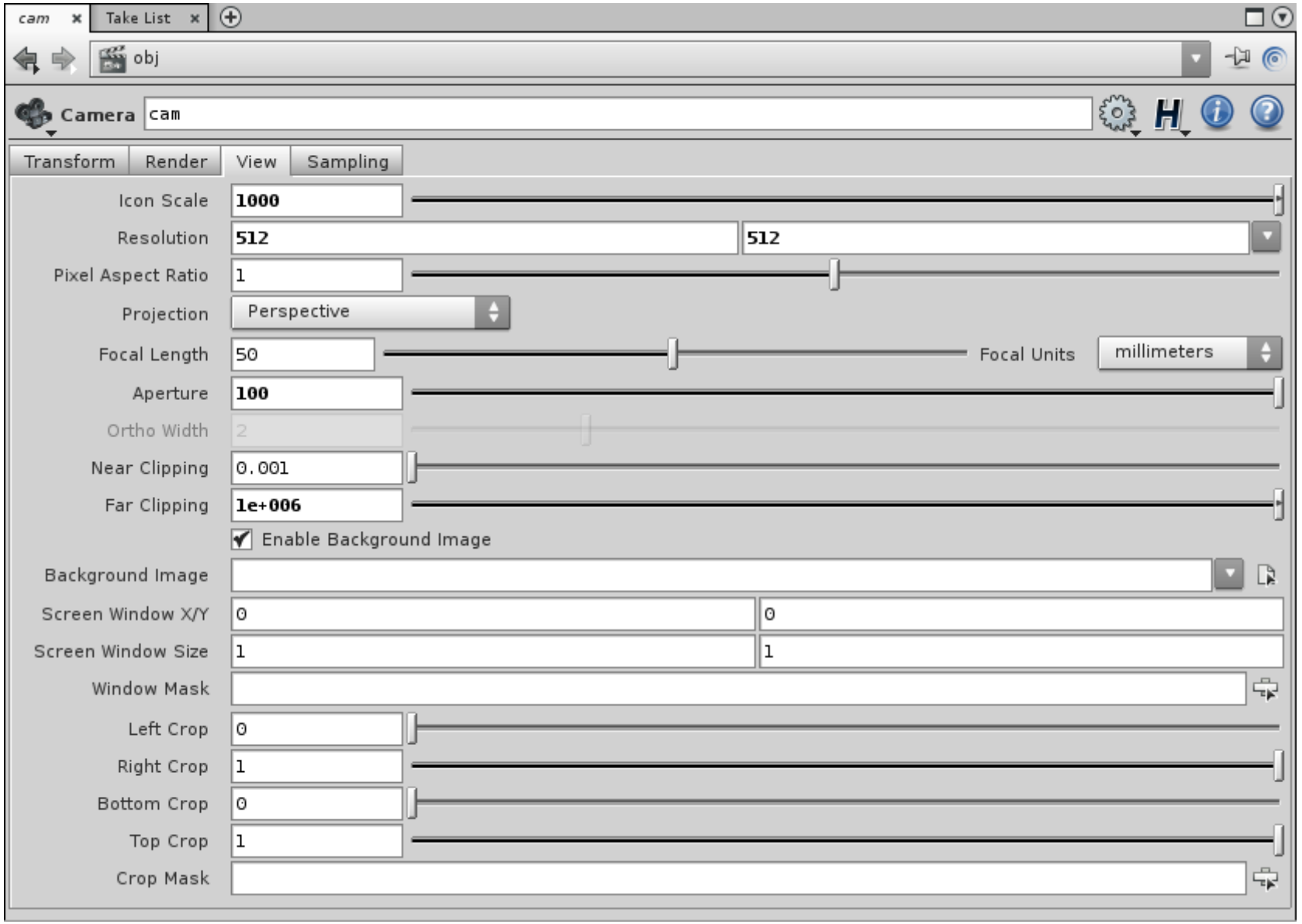


Now you can see color of flow.

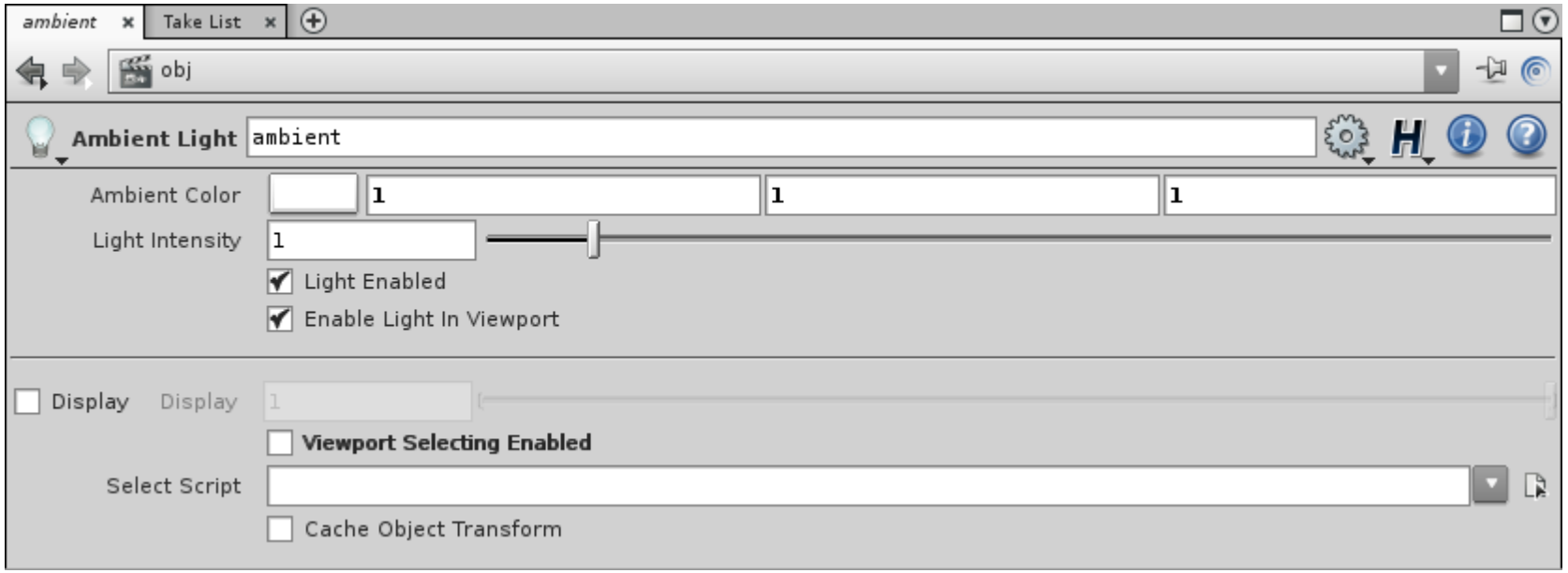


Return to **obj** network by clicking this button  on top of **Network** view.

TAB -> Camera



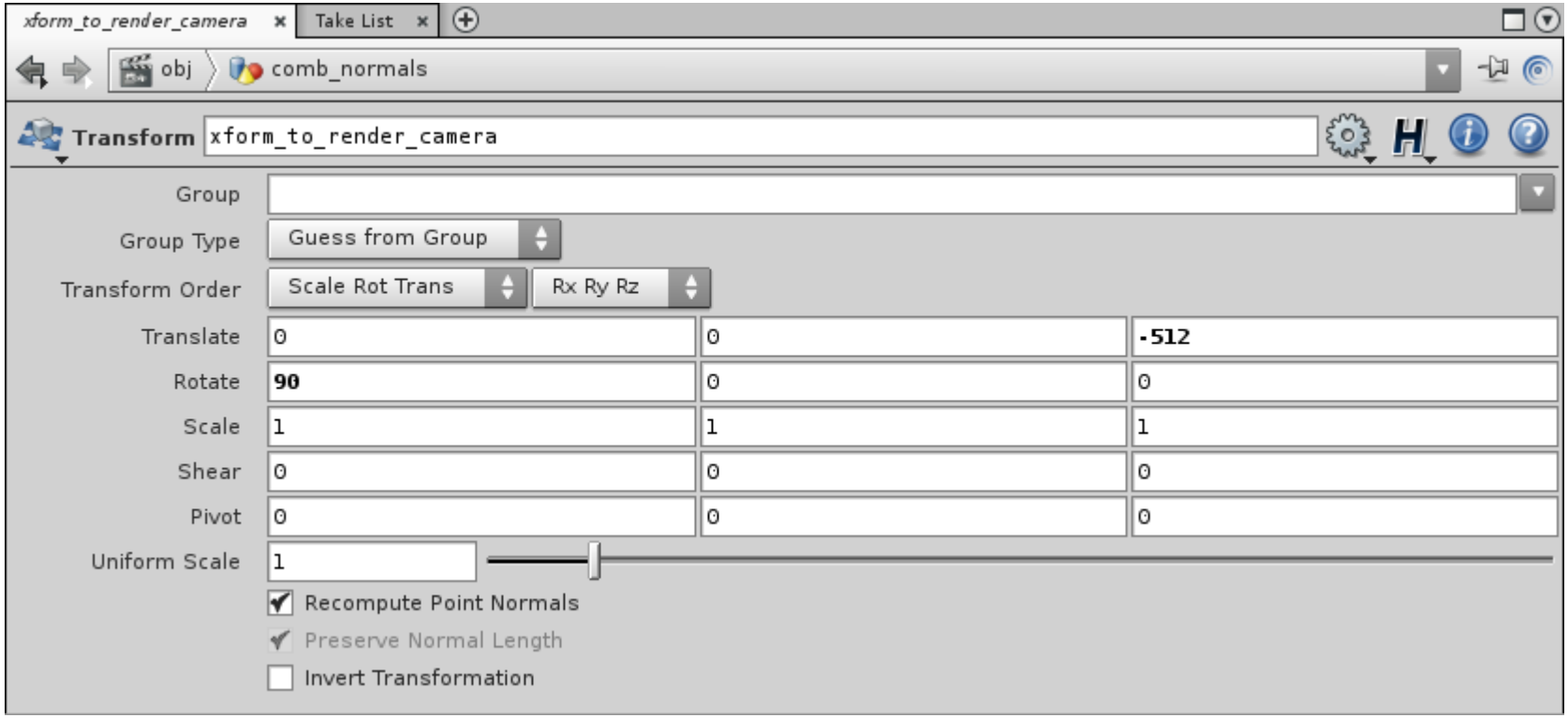
TAB -> **Ambient Light**



As you can see, this is simple camera and ambient light objects, needed for render flow texture.

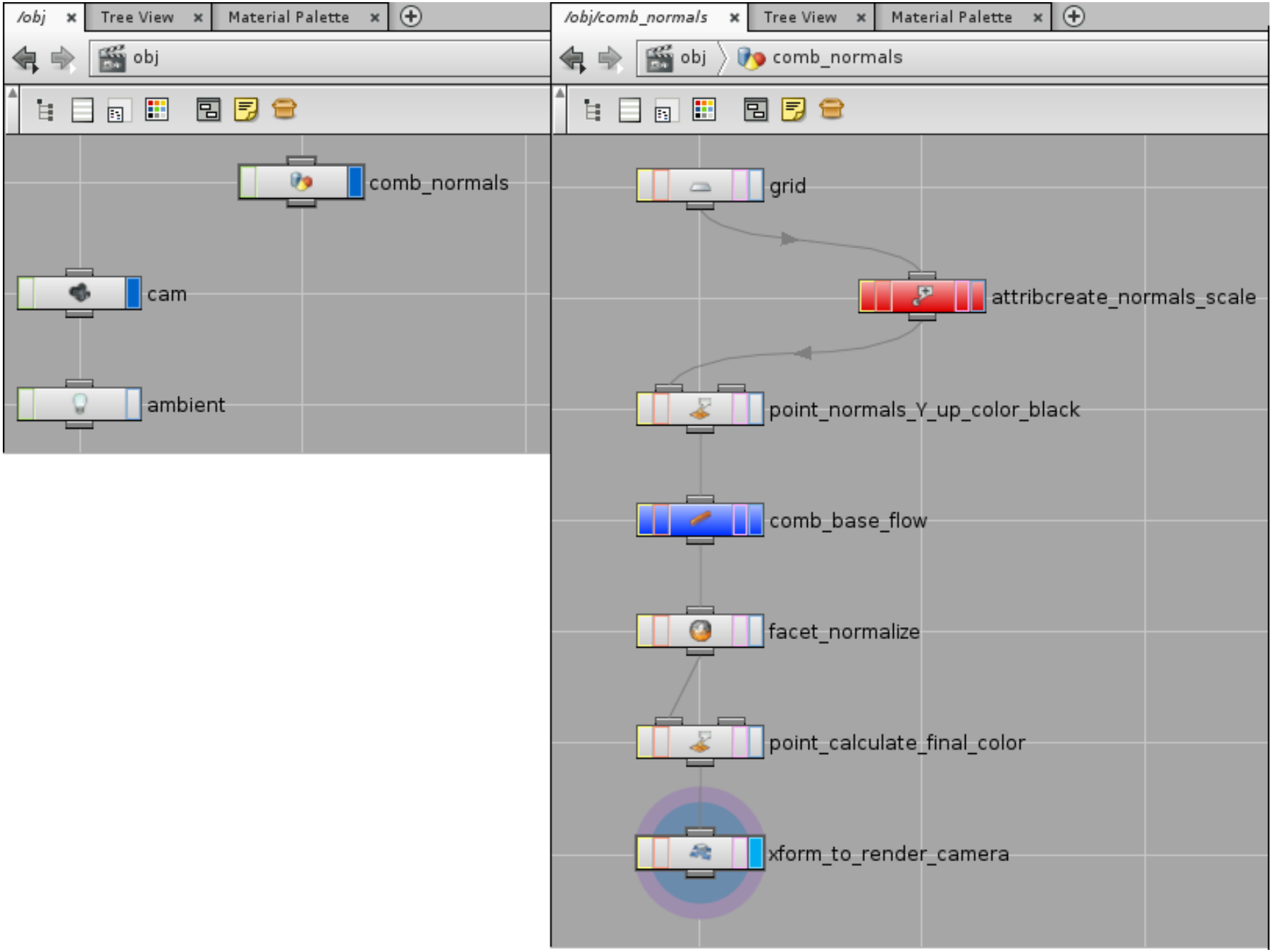
Open **Grid** object, and under last **Point** node (calculate color) add

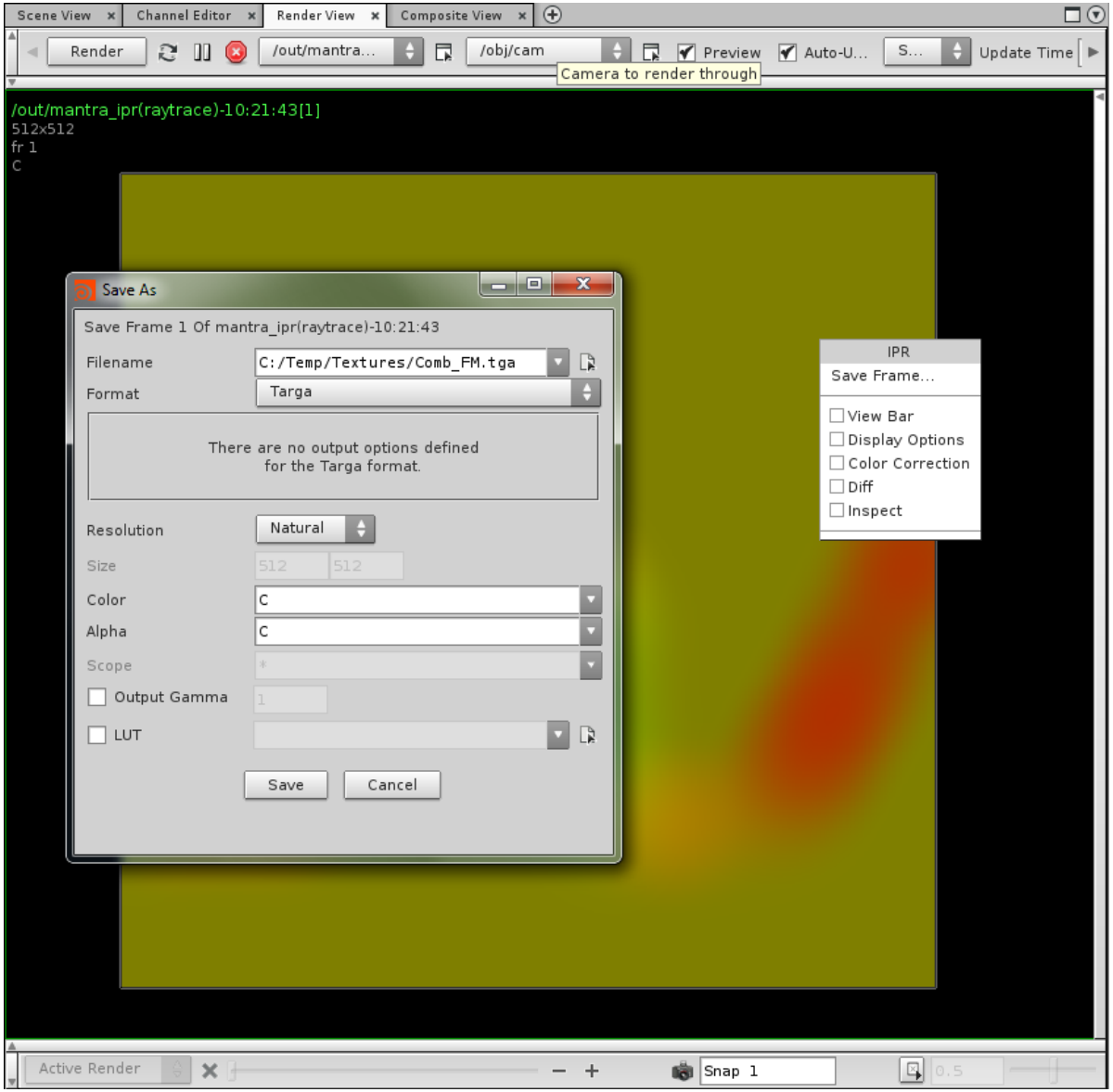
TAB -> **Transform** 



Now your grid transformed to desired render position.

Final network.





On top of **Viewport** you can find “Render View” tab. Click on it.

Select your camera in “Camera to render through”.

Press “Render” button.

When render finished, right click on final image and press “Save Frame...”.

In “Save As” window select your file extention, name and place to save.

(!) <u>Limitation – you can’t change tessellation of your grid after you comb normals.</u>
It will change points order, and combing will be broken.

Same limitation for **Paint** tool 

Calculate normals from deformation

Repeat all steps to “point_normals_Y_up_color_black”.

TAB -> **Metaball** 

This is a simple metaball, which, in this case, used as spherical field.

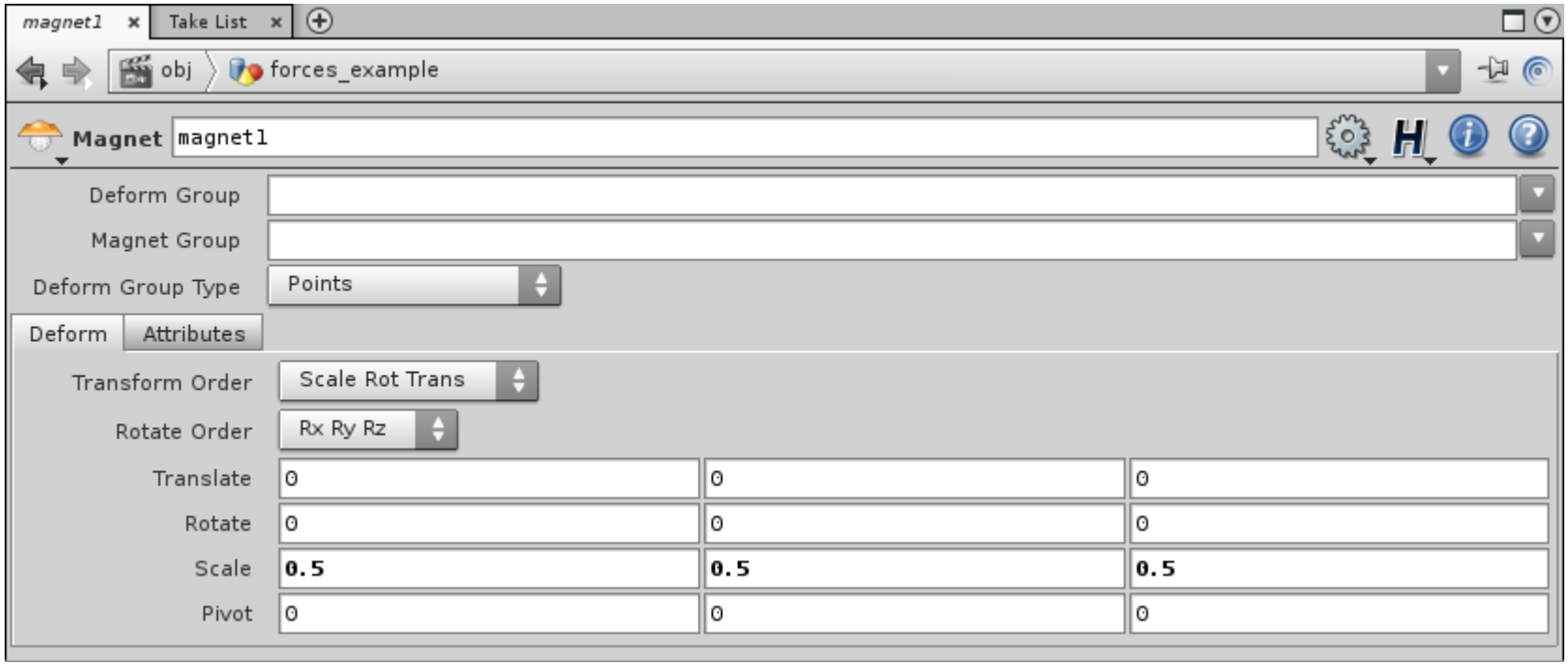
TAB -> **Magnet** 

Magnet can deform something, using provided attributes, for example spherical field.

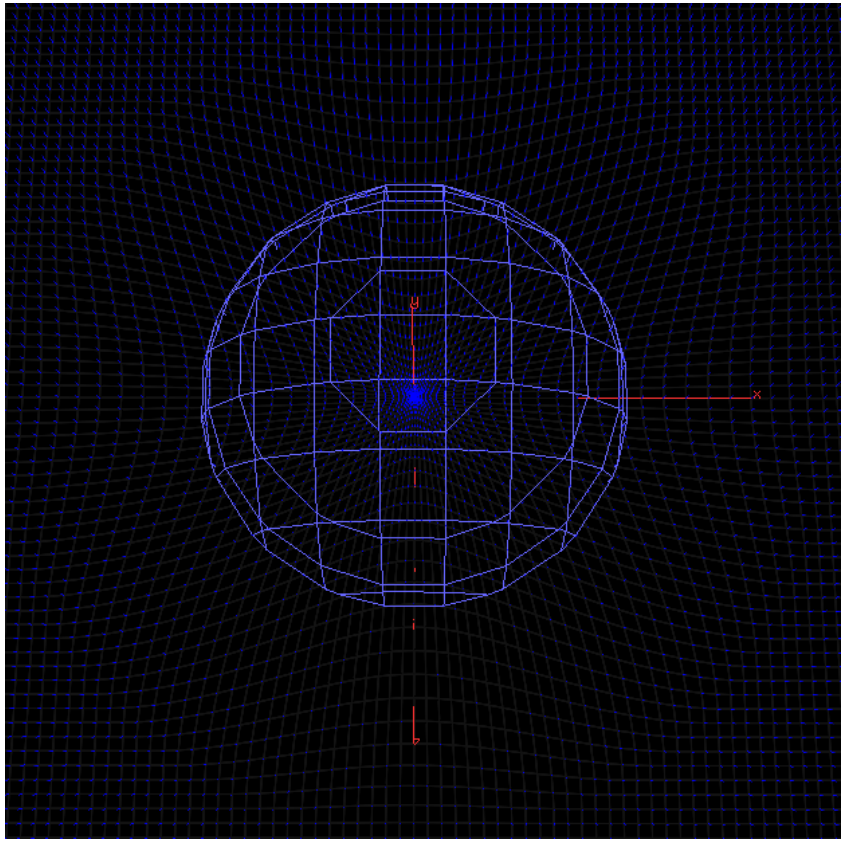
Connect your **Grid** (point_normals_Y_up_color_black), with 1st (left) **Magnet** input, and **Metaball** with it's 2nd (right) input.

Change only Scale attribute in **Magnet**.

Scale = 0.5, 0.5, 0.5.



Now your **Magnet** must affect points positions, and you will see this



Using simple mathematic operations between old and new points positions, you can get effects of upwellings, downwellings and vortices with clockwise and counterclockwise directions.

TAB -> **Point**, for new normals.

Connect “point_normals_Y_up_color_black” with 1st **Point** input, and **Magnet** with it’s 2nd input.

Normal -> Add Normal

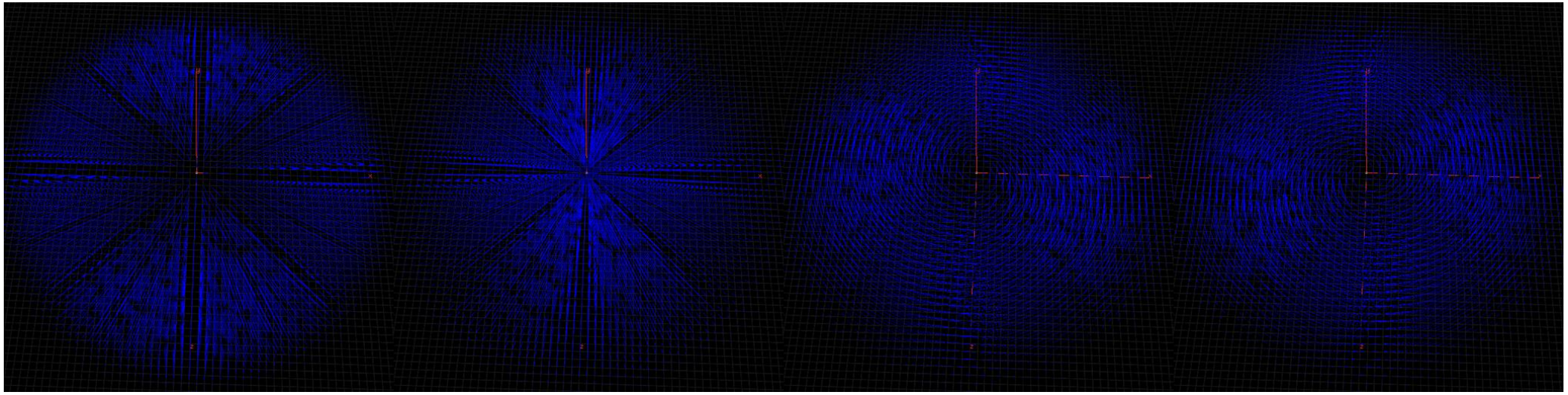
Upwellings = $TX-$TX2, 0, $TZ-$TZ2.

Downwellings = -($TX-$TX2), 0, -($TZ-$TZ2).

Vortices_CW =
(cross(vector3($TX-$TX2, 0, $TZ-$TZ2), vector3($NX, $NY, $NZ))/$NS) [0],
0,
(cross(vector3($TX-$TX2, 0, $TZ-$TZ2), vector3($NX, $NY, $NZ))/$NS) [2].

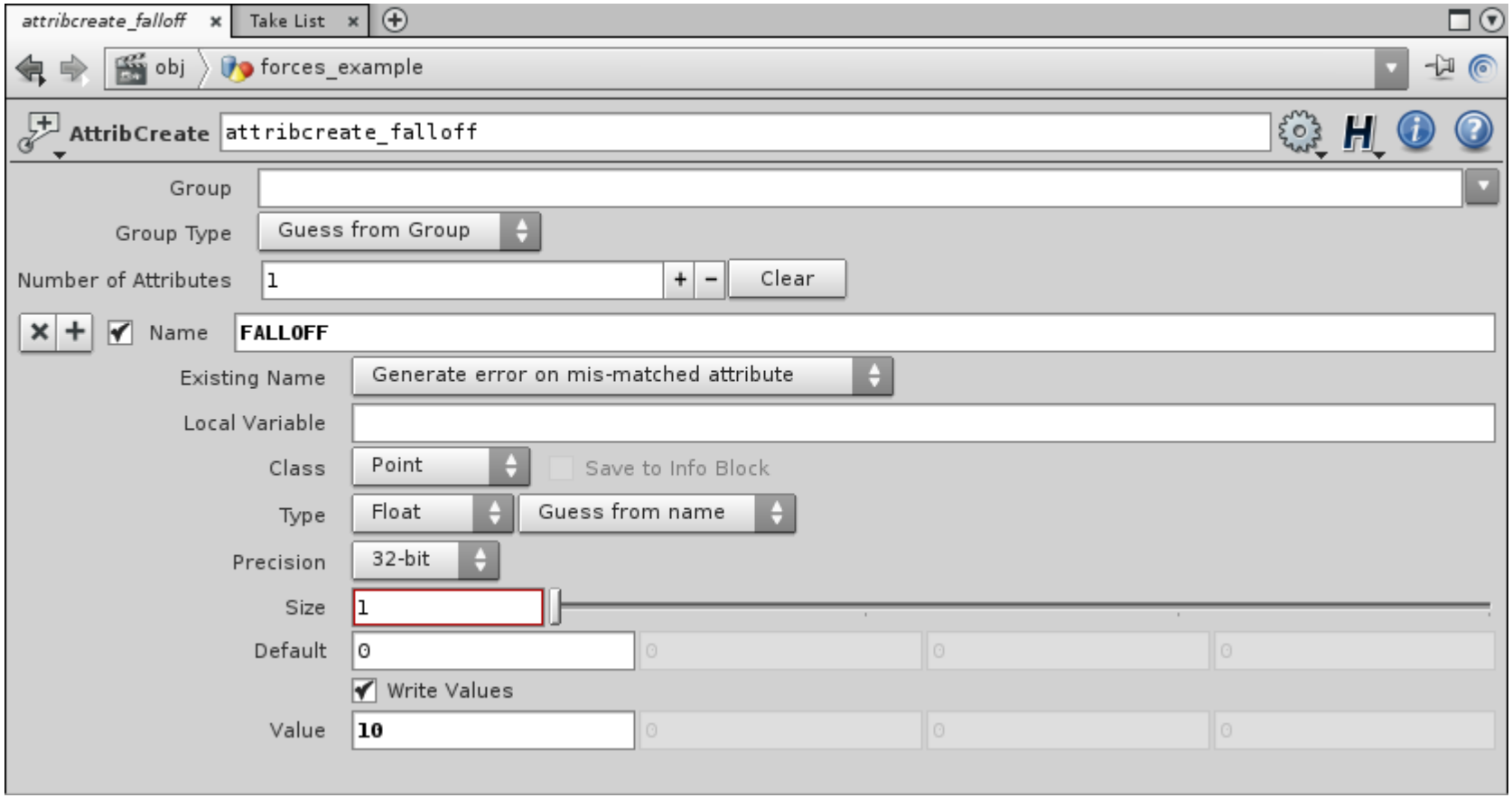
Vortices_CCW =
(-cross(vector3($TX-$TX2, 0, $TZ-$TZ2), vector3($NX, $NY, $NZ))/$NS)[0],
0,
(-cross(vector3($TX-$TX2, 0, $TZ-$TZ2), vector3($NX, $NY, $NZ))/$NS)[2].

Results:



As you can see, new vectors shorter in center, longer in middle, and again shorter on edge.
This is not correct. Vectors must be longer in center and fall off to edge. It will be fixed with normalizing.

TAB -> **AttribCreate**



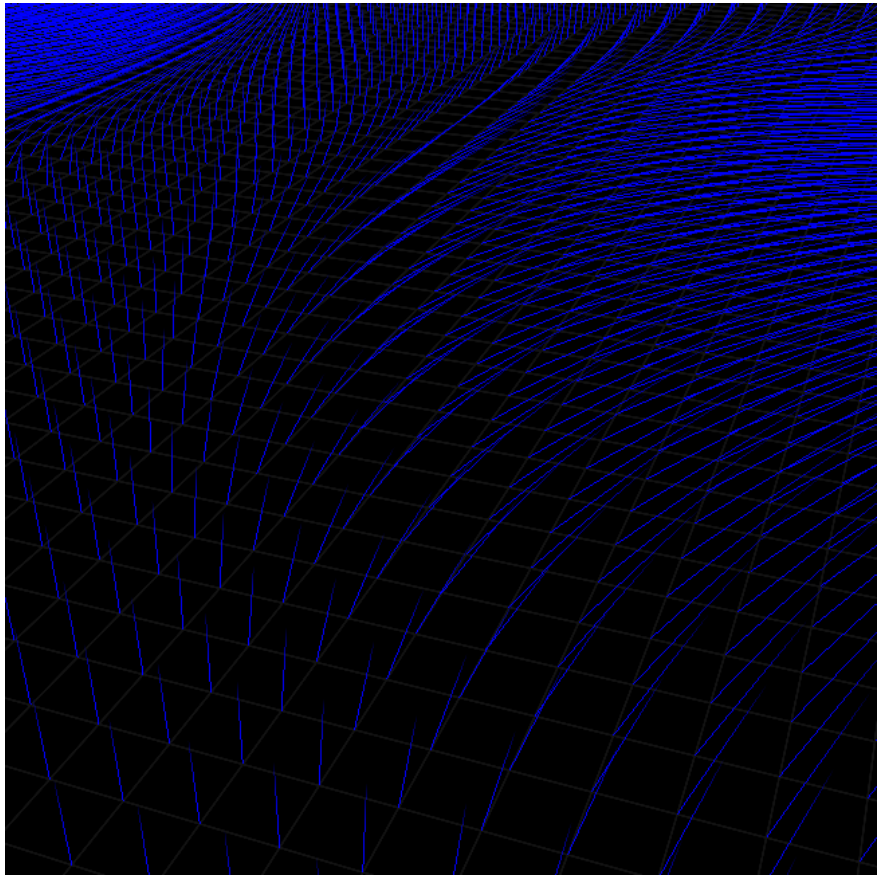
This attribute will be used for vectors alignment to up (Y) axis.

TAB -> **Point**

Normal -> Add Normal = $NX, $FALLOFF, $NZ.

Connect Falloff attribute with 1st point input.

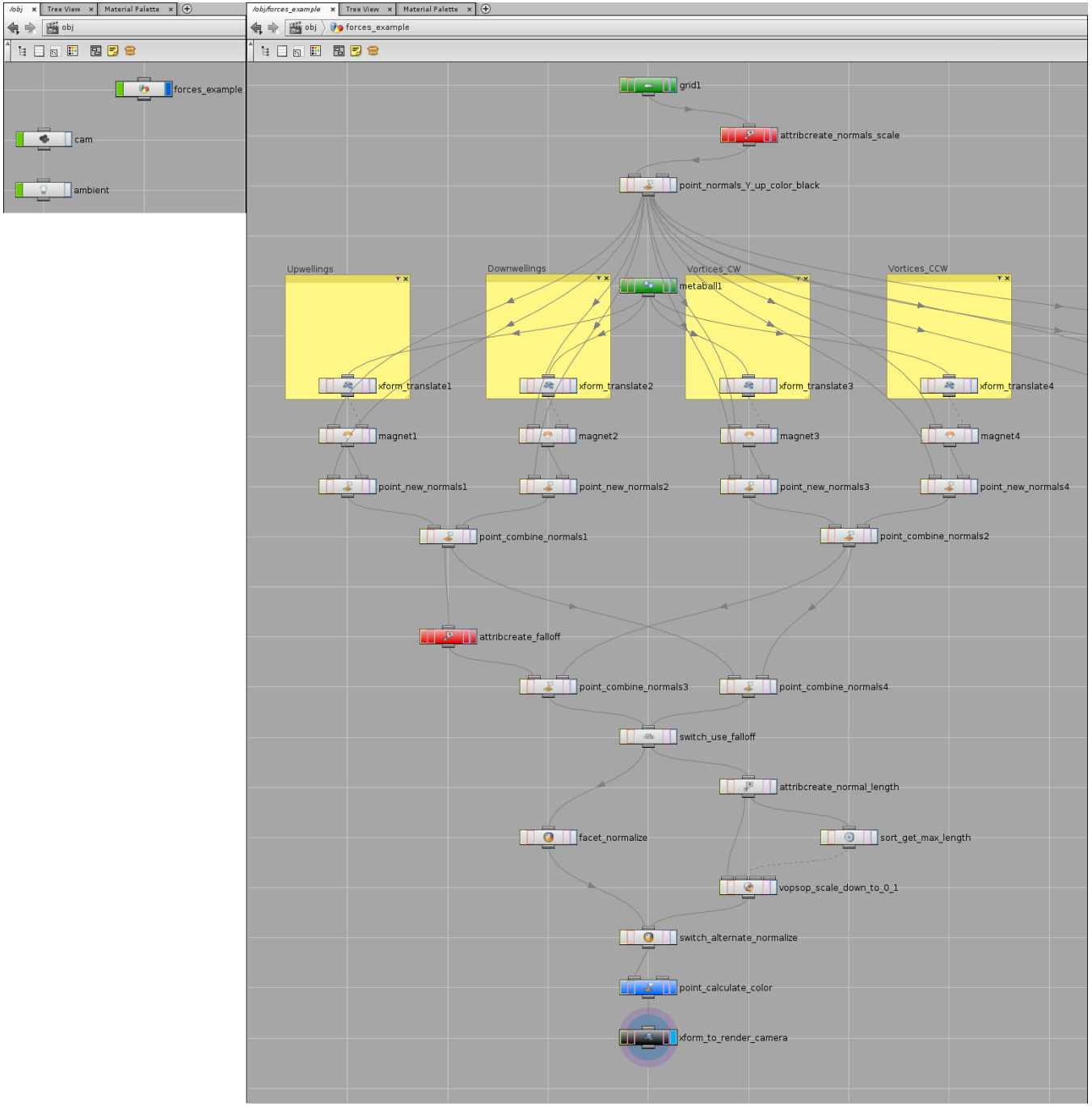
After calculations from deformation, your normals was parallel to grid plane.
If you add some value to $NY field of **Point**, normals begin to raise.



If you normalize vectors while it parallel to grid, you will get effects without falloff on edge.
But, if your normals have some $NY value, it's $NX and $NZ values on edge can be <1.

Finally add **Facet** "normalize", **Point** "calculate color" and **Transform** "to render camera".

Final network.

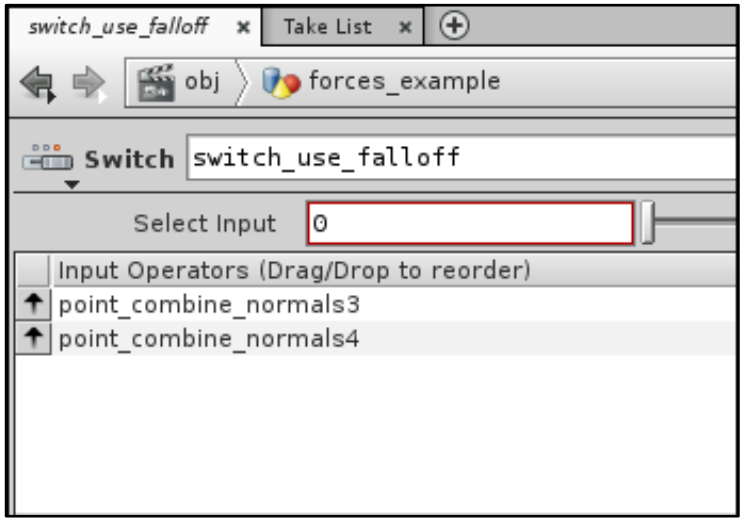


I create 4 branches with different transforms of **Metaball** to show all effects.

Nodes “point_combine_normals” simply add one normals to another.
Normal -> Add Normal = $NX+$NX2, 0, $NZ+$NZ2.

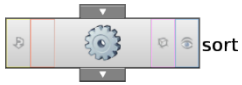
Here you can see new node **Switch** 

This is a simple switch between branches. In it’s properties you can see connected nodes and change its order.



Also here are “alternate normalize” branch.

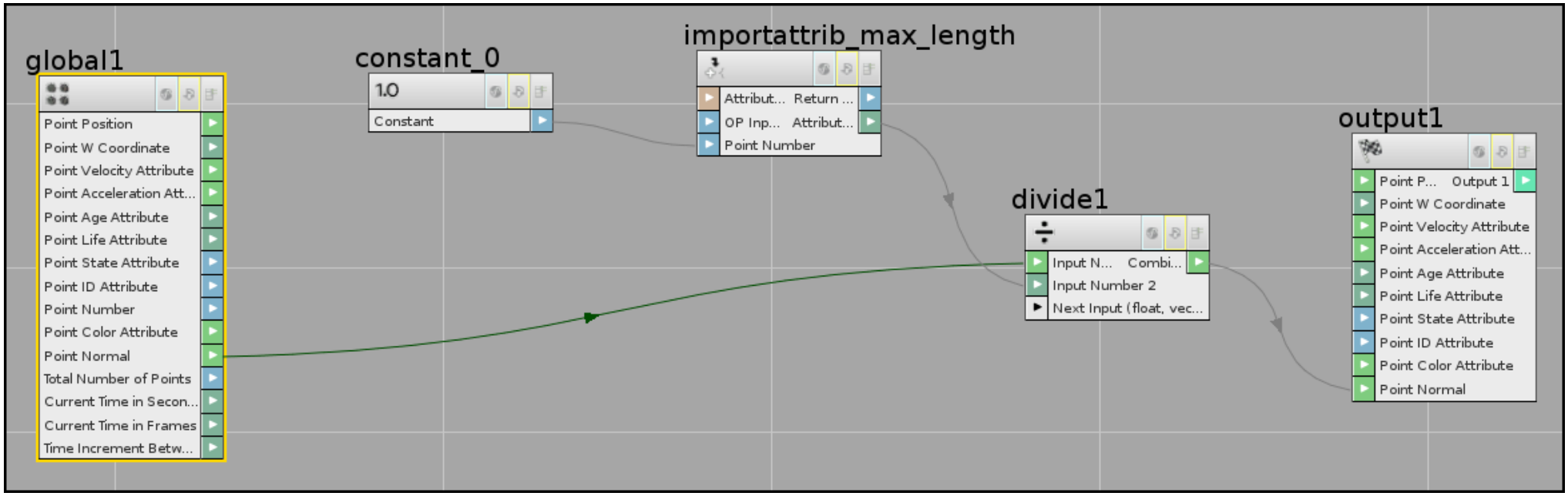
How it works.
Node “attribcreate_normal_length” get lengths of all points normals.
Name = NORMAL_LENGTH.
Local Name = NL.
Class = Point.
Type = Float.
Value = length($NX, $NY, $NZ).

Node **Sort** 

sorting array of this values, by expression -$NL. So, it gets maximum normal length and place it in 0 index of array.

Then, node **VOPSOP** 

divide all normals on that maximum length. Open it to see.



Global – data from 1st input of **VOPSOP**. You need it's Point Normals.

Constant – simple constant.
Type = Integer.
Value = 0.

Import Attribute – operator, which gets specified value from specified **VOPSOP** input.
Signature = Float.
Attribute = NORMAL_LENGTH.
OP input index = 1 (2nd input of **VOPSOP**).

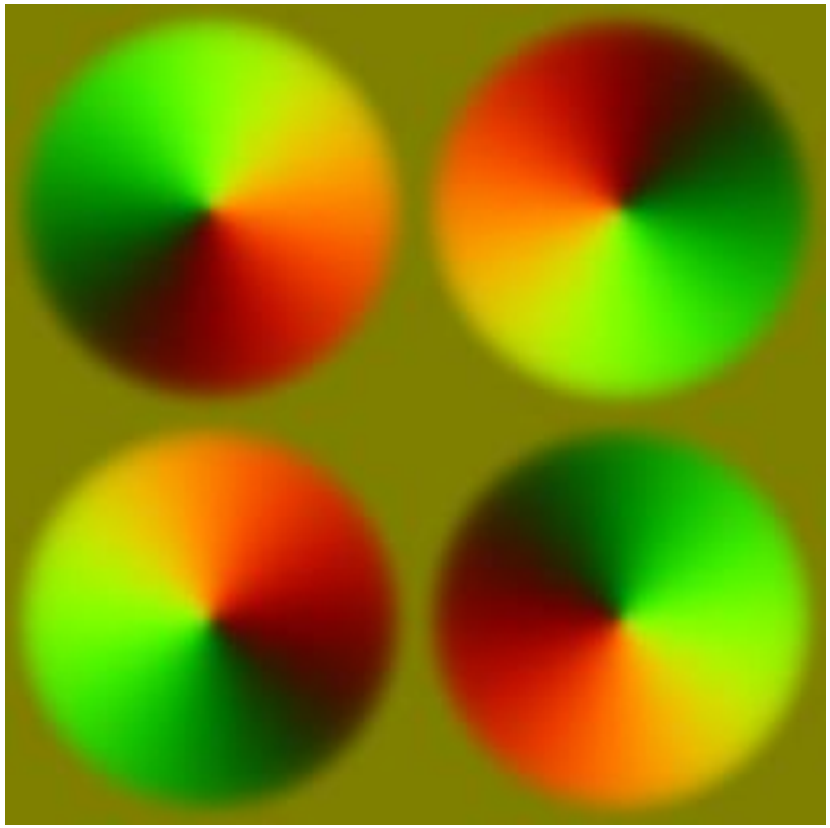
So, points normals will be divided on value in 0 index of sorted array with normals lengths, on maximum length.

And return it to **VOPSOP** Point Normal output.

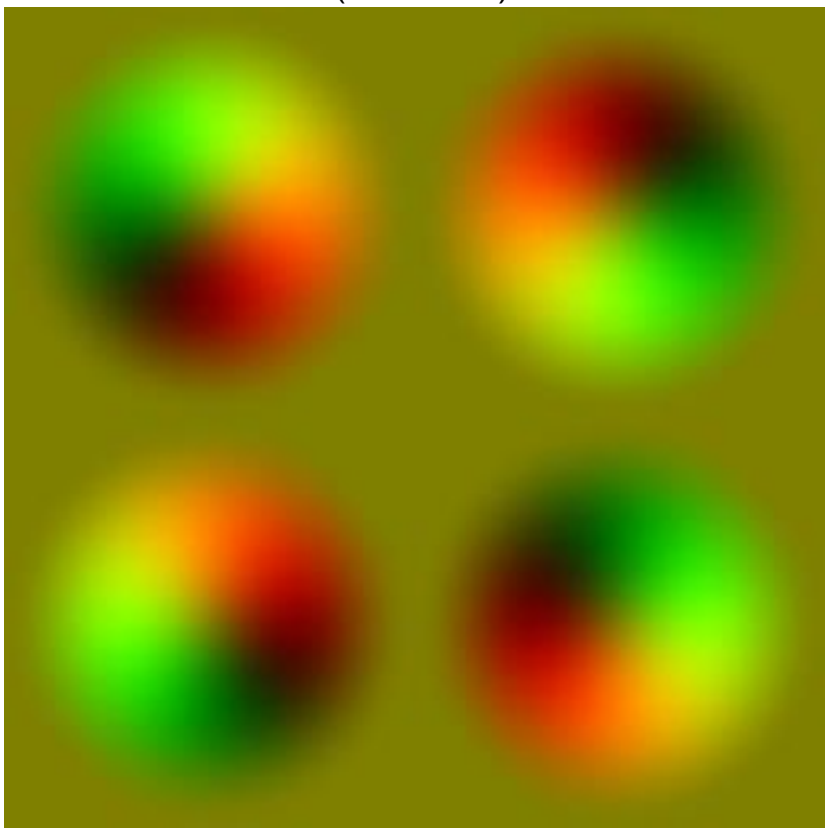
As result, you will get normals with length from 0 to 1, and now you don’t need any other normalization.

But, it different from simple normalization with **Facet**.

Facet



Alternate normalization (scale down)



In this alternative normalization you get actual values of vectors – shorter in center, longer in middle, shorter on edge.

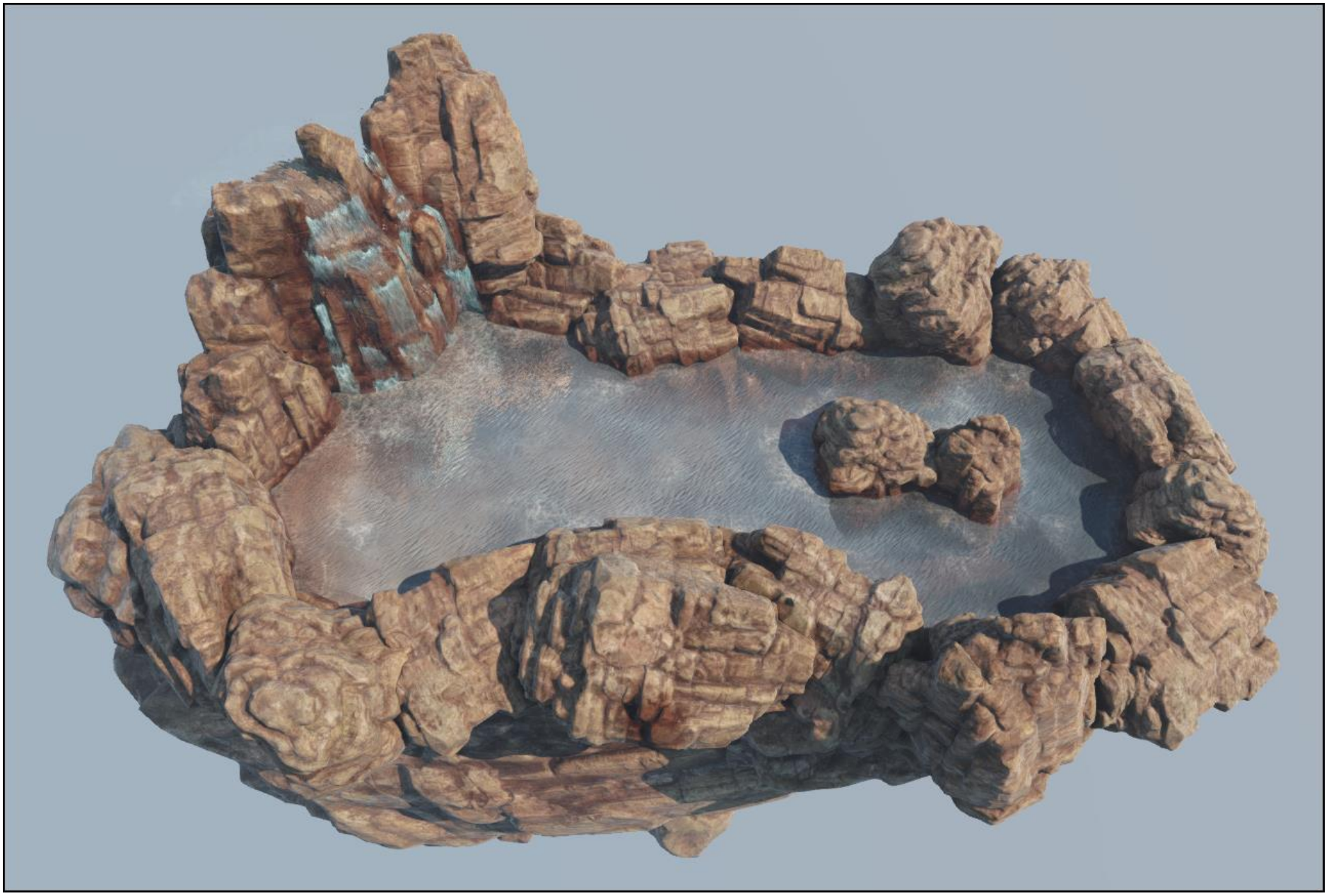
So, that's enough information for now, to think about.

There is one method I left for later – transfer attributes from one object to another.

Let's go to next chapter, and look at specific example of using **Houdini** to create flow maps for real level.

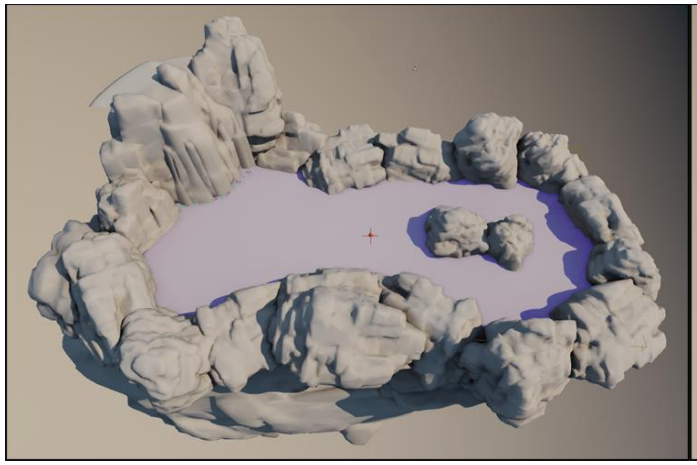
Houdini main setup

For example, I build simple map with rocks, waterfall and water surface.

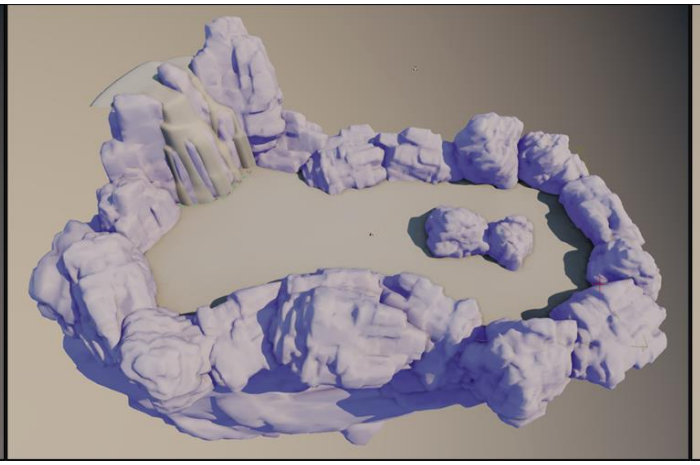


Separately export water, rocks and waterfall as *.obj files.

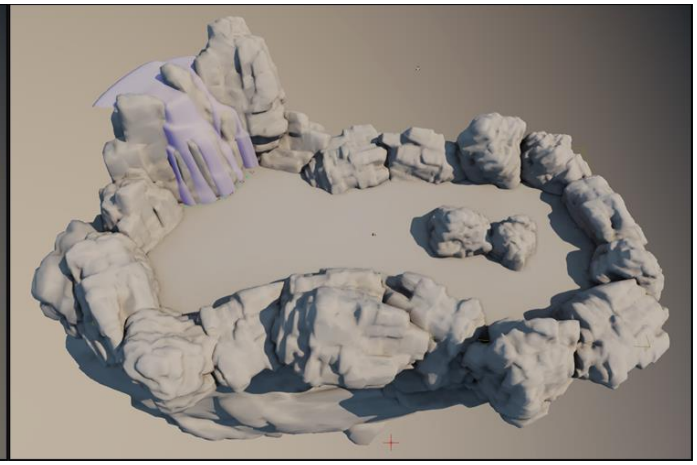
FlowTest_Water.obj



FlowTest_World.obj



FlowTest_Waterfall.obj



Import it into **Houdini** (*File -> Import -> Geometry*).

Also, add **Camera** and **Ambient Light**.
Hide all objects by clicking on it's "display" icons.

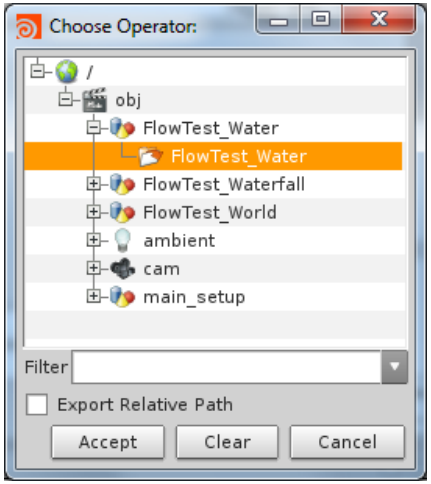
TAB -> **Geometry**

It will be main network. Open it.

TAB -> **Object Merge**

In it's properties find "Object1" field, click on "Open floating operator chooser"  and select water mesh

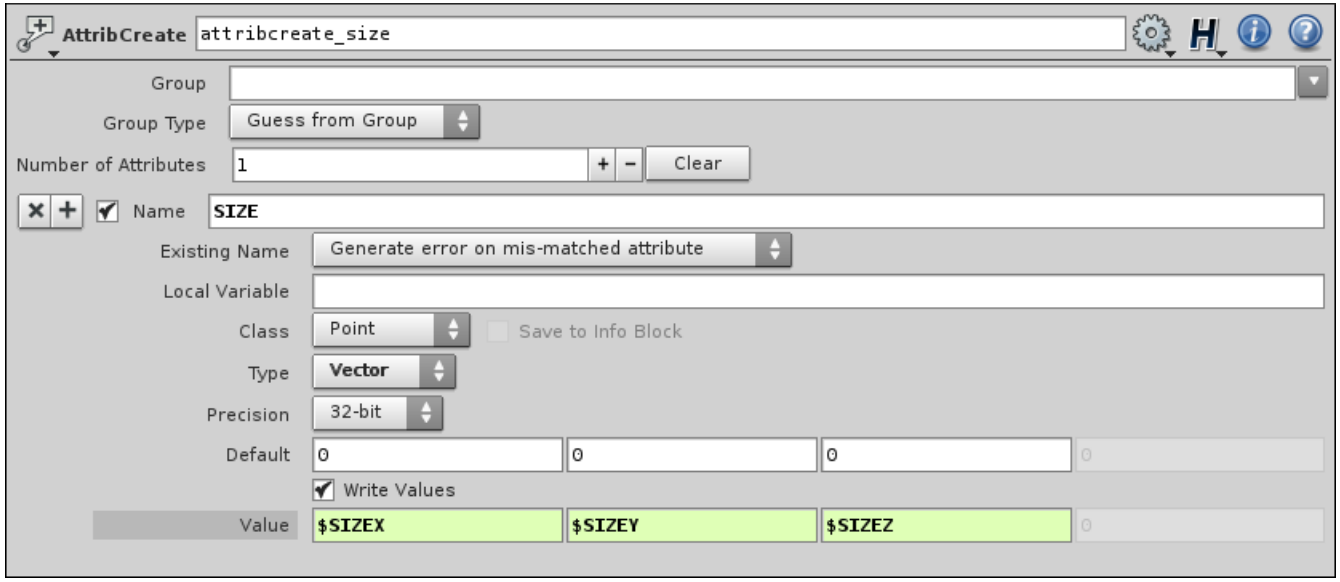
Repeat for World and Waterfall for merging it with this network.



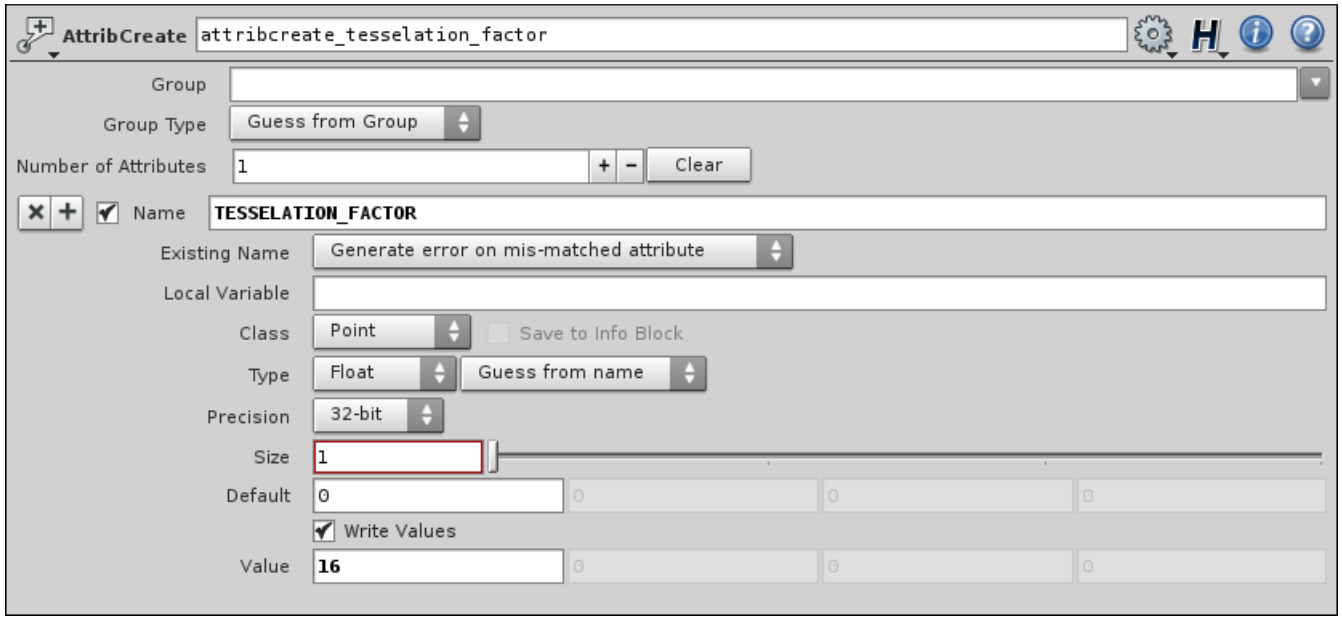
What do you need now?

Step 1 – Replace water plane with tessellated grid

“attribcreate_size” gets and keeps size of water plane.

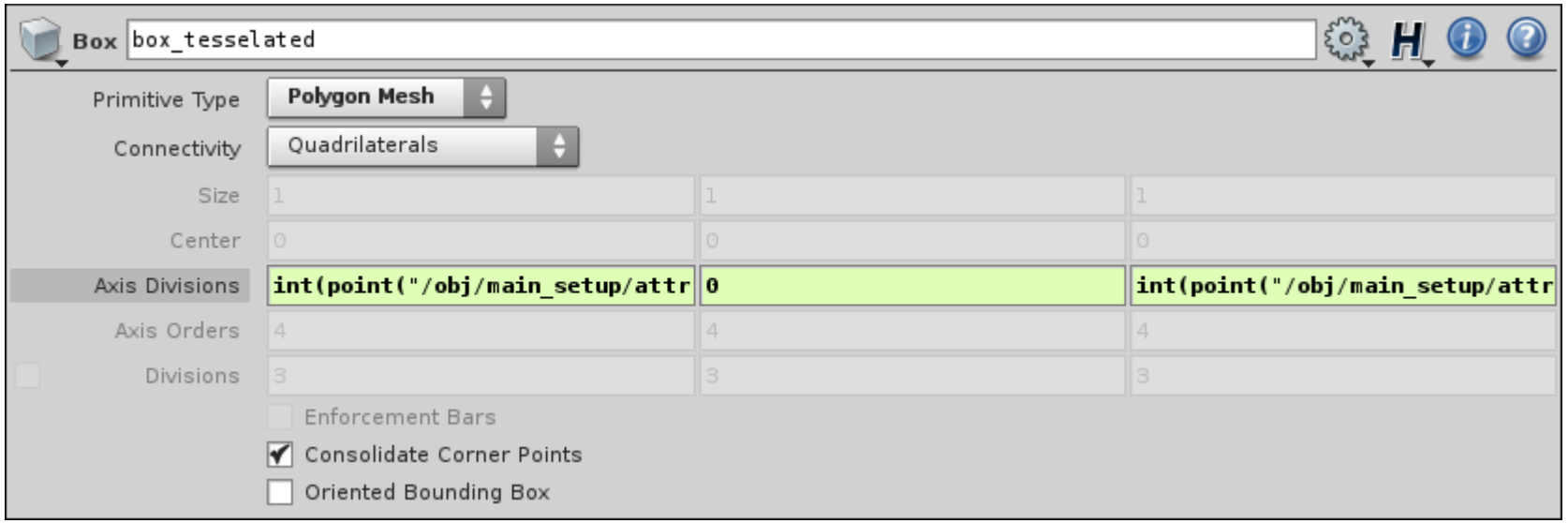


“attribcreate_tesselation_factor” provides new attribute – length of new polygon’s edge.
So, one quad will be created on Value size.



“box_tesselated” – simple box object. It uses $SIZE and $TESSELATION_FACTOR for Size and Axis Divisions parameters.

Box 



Axis Divisions = int(point("/obj/main_setup/attribcreate_size", 0, "SIZE", 0)/ch("/obj/main_setup/attribcreate_tesselation_factor/value1v1")) + 1,
0,
int(point("/obj/main_setup/attribcreate_size", 0, "SIZE", 2)/ch("/obj/main_setup/attribcreate_tesselation_factor/value1v1")) + 1.

Formulas means – get $SIZE(X,Z) of “attribcreate_size” node, and divide it on Value of “attribcreate_tesselation_factor” plus 1.

If you click on “Axis Divisions” label, you will see current values – 129, 0, 257.

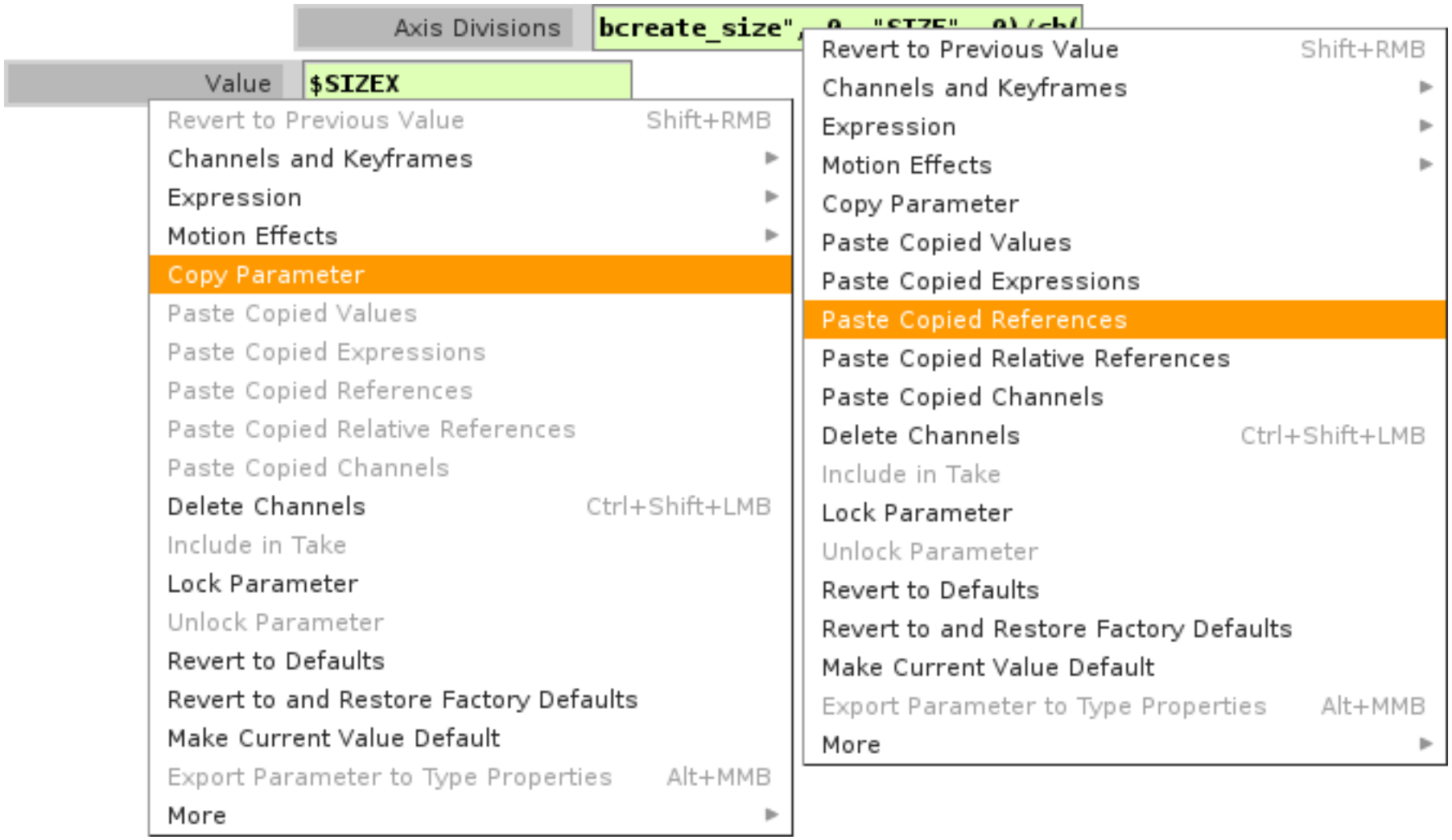
That water plane have size 2048*4096 units.

2048/16 + 1 = 129.
4096/16 + 1 = 257.

Looks like everything is fine.

You can get path of attribute by right click on it's label or in it's field, and select "Copy Parameter".

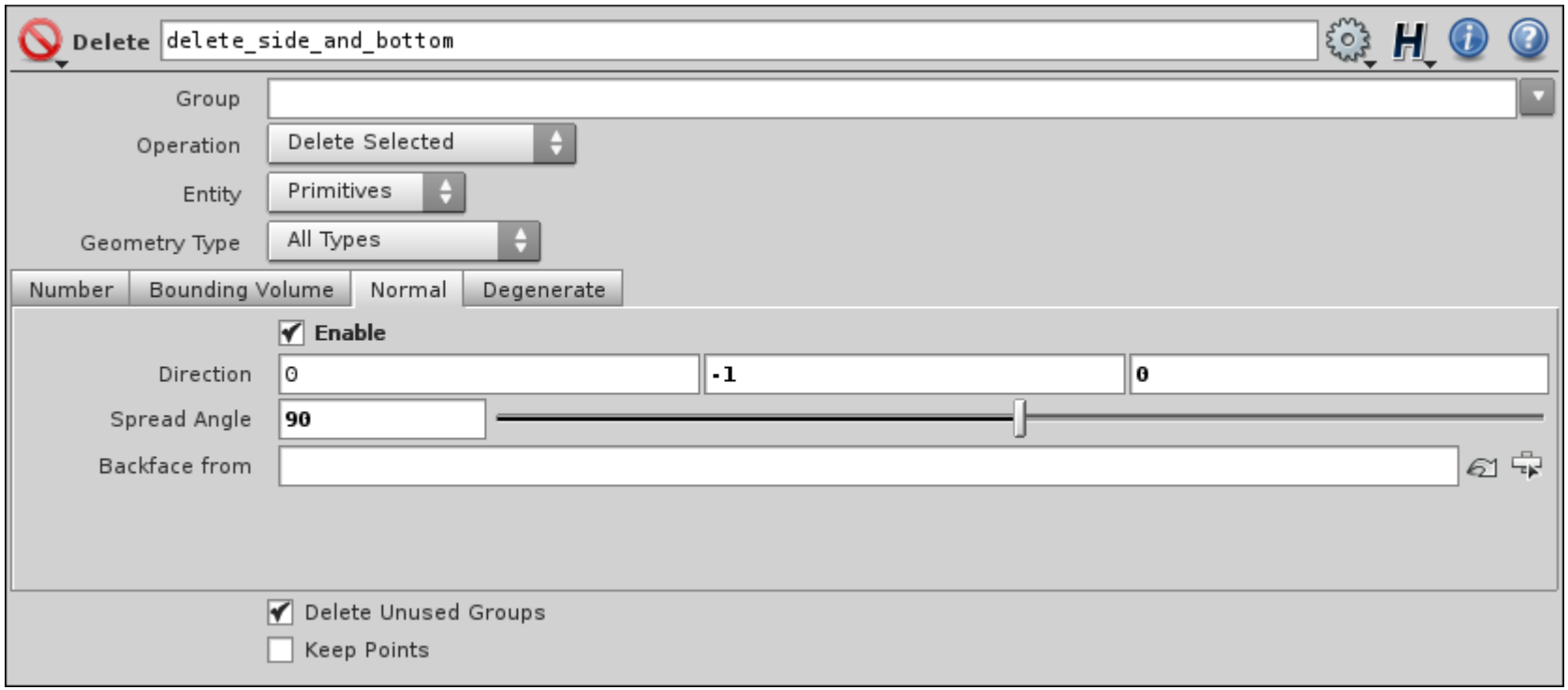
To paste it somewhere – right click on it's field -> "Paste Copied References".



“delete_side_and_bottom” – **Delete** operator 

It will delete side and bottom sides of new box, and leave top side, which will be your grid.

Uncheck “Enable” in “Number” tab.
Go to “Normal” tab and check “Enable”.
Operation = Delete Selected.
Direction = 0, -1, 0.
Spread Angle = 90.

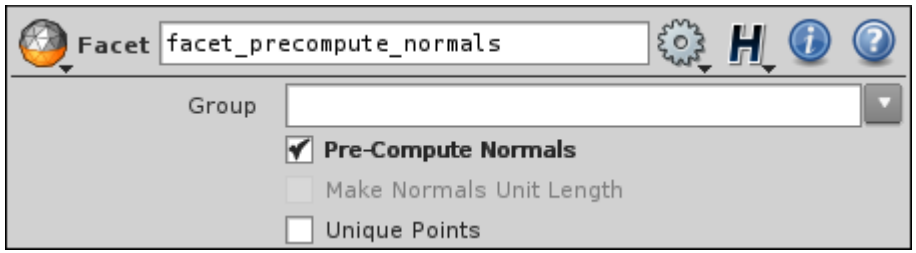


Another way:

Operation = Delete Non-Selected.
Direction = 0, 1, 0.
Spread Angle = 90.

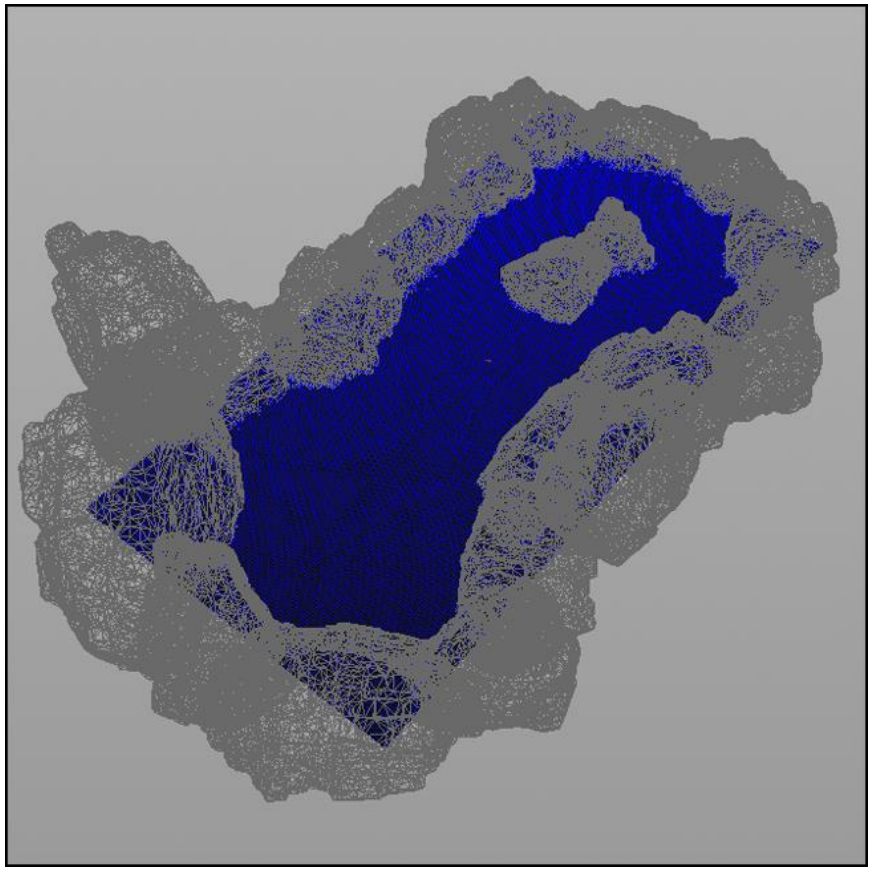
It’s the same.

Facet -> Pre-Compute Normals, just in case.



“attribcreate_normals_scale” and “point_normals_Y_up_color_black” you already know.

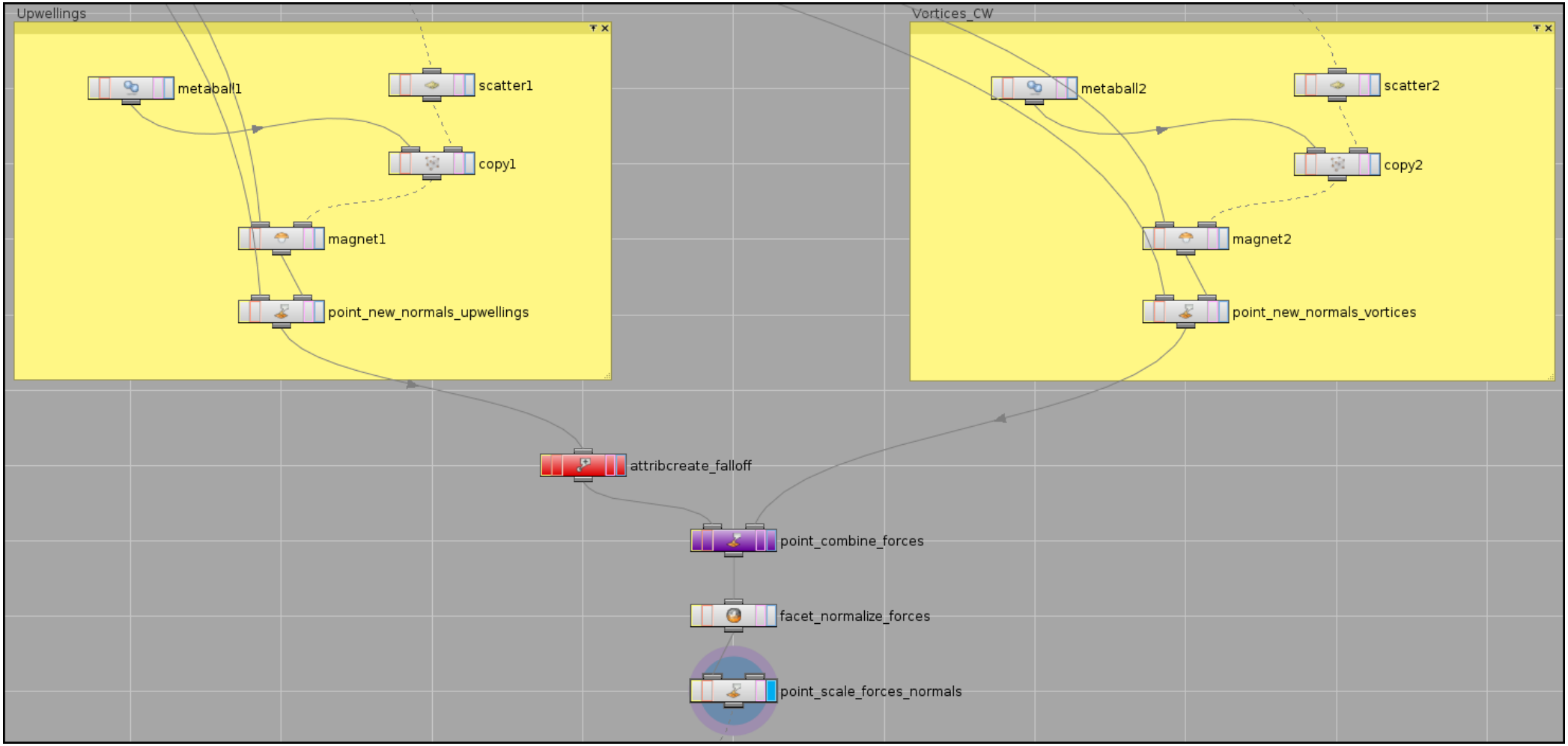
Finally you get new grid.



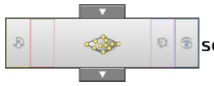
Click on “template” icon  on **Object Merge** node of World mesh, to see it’s template preview.

Step 2 – Creating forces, or “turbulent surface”

In this case, I use only upwellings and clockwise vortices.

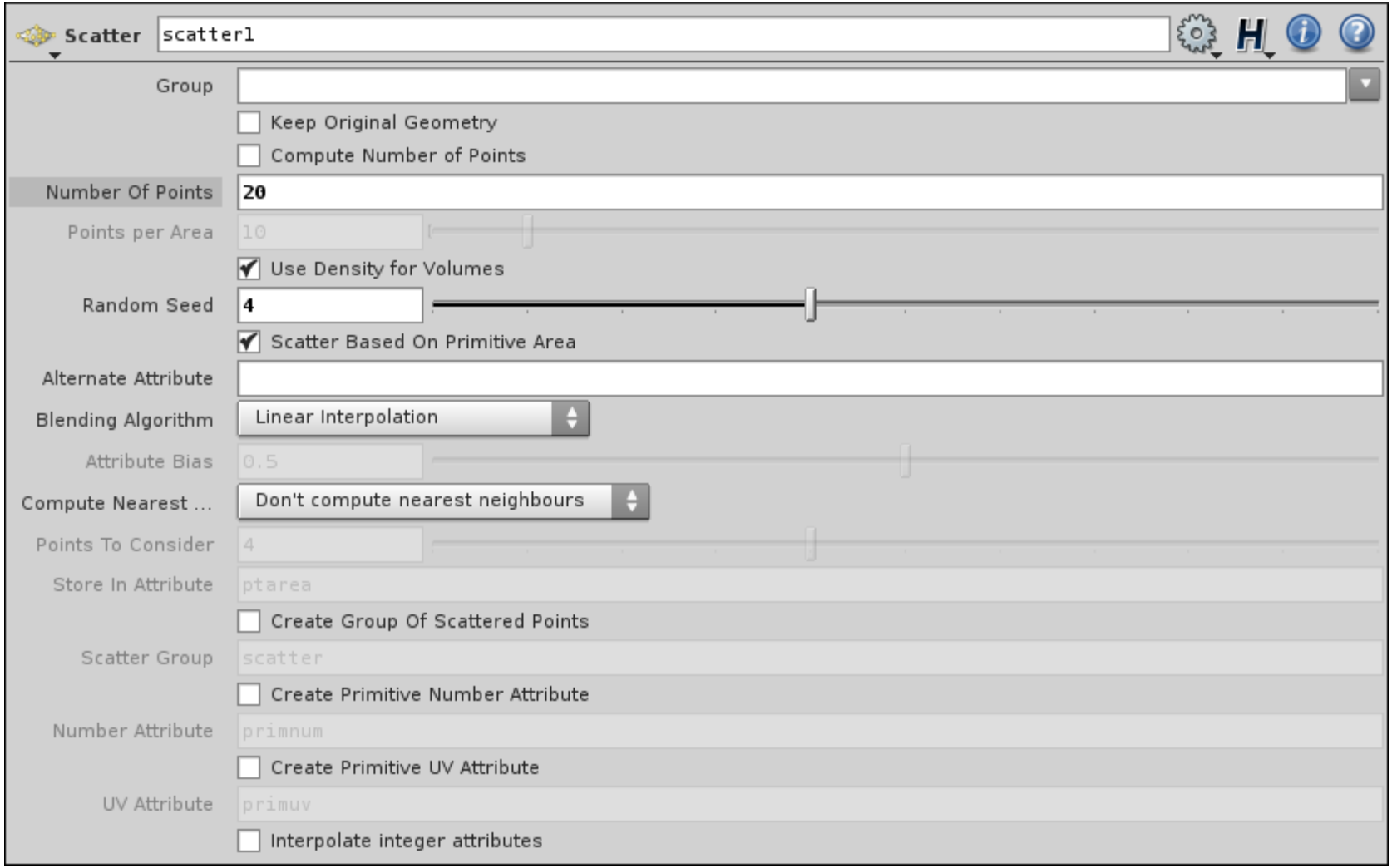


Metaball, as in previous chapter, use Mode “Threshold Radius” and, for upwellings, have Size = 250, 250, 250, for vortices = 150, 150, 150.

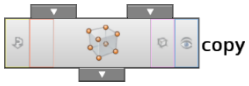
Scatter 

can scatter points on specified surface.

“point_normals_Y_up_color_black” connected with Scatter input.



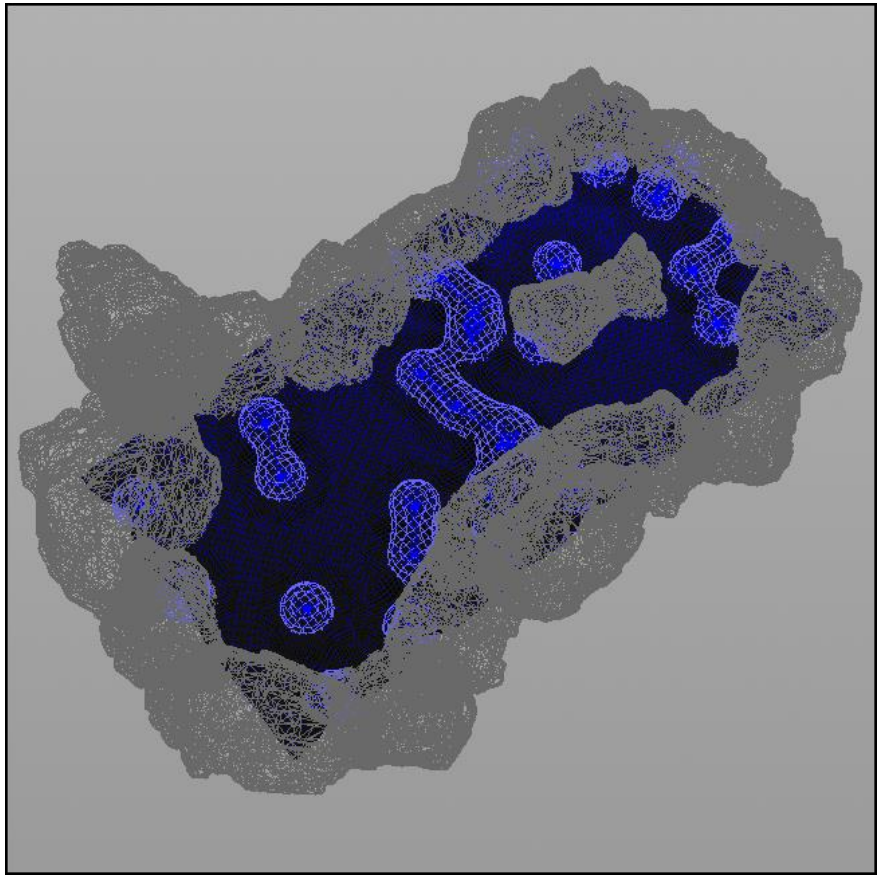
20 points for upwellings, 30 for vortices.

Copy 

copy metaballs to each point.

Magnet's Scale for upwellings = 0.5, for vortices = 0.25.

Toggle "display" of any **Magnet**, and select it.
You can see how metaballs now placed on water surface.

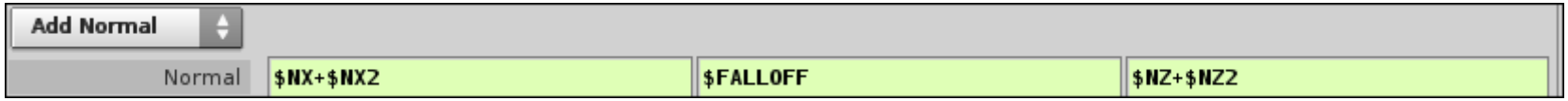


Play with **Scatter**'s "Random Seed" value to get desired result.

Point "new normals" you already know.
Point and **Magnet** also connected with water grid.

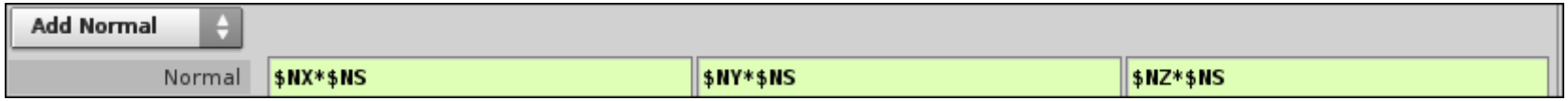
"attribcreate_falloff" also known.

“point_combine_forces”

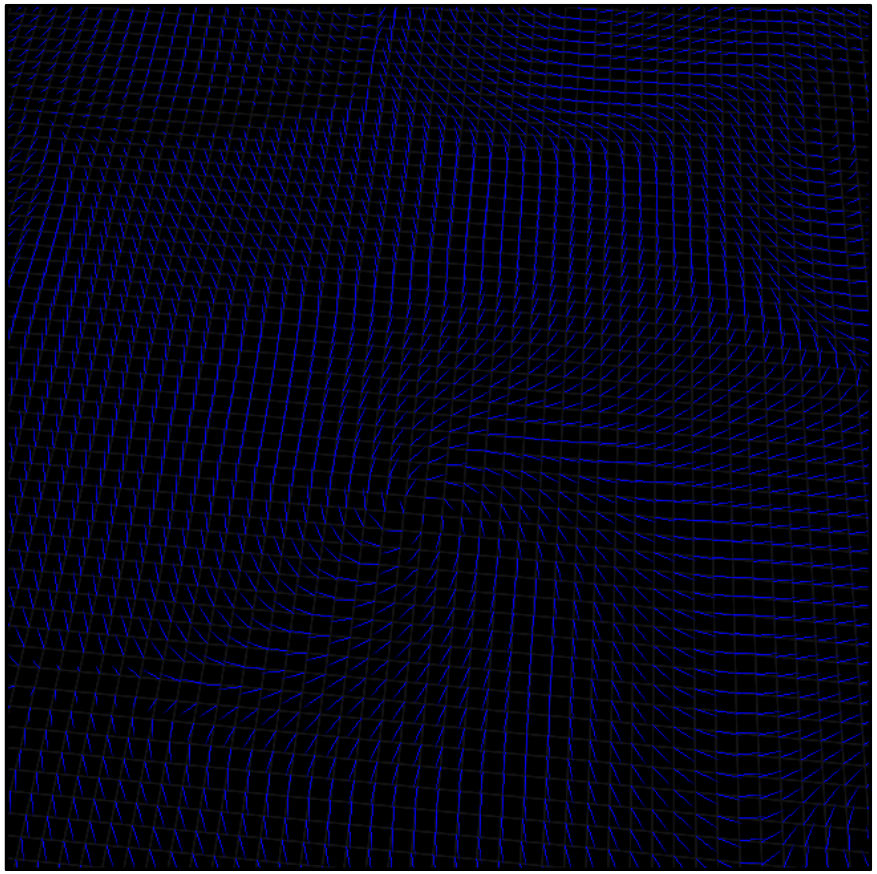


“facet_normalize_forces” -> Make Normals Unit Length.

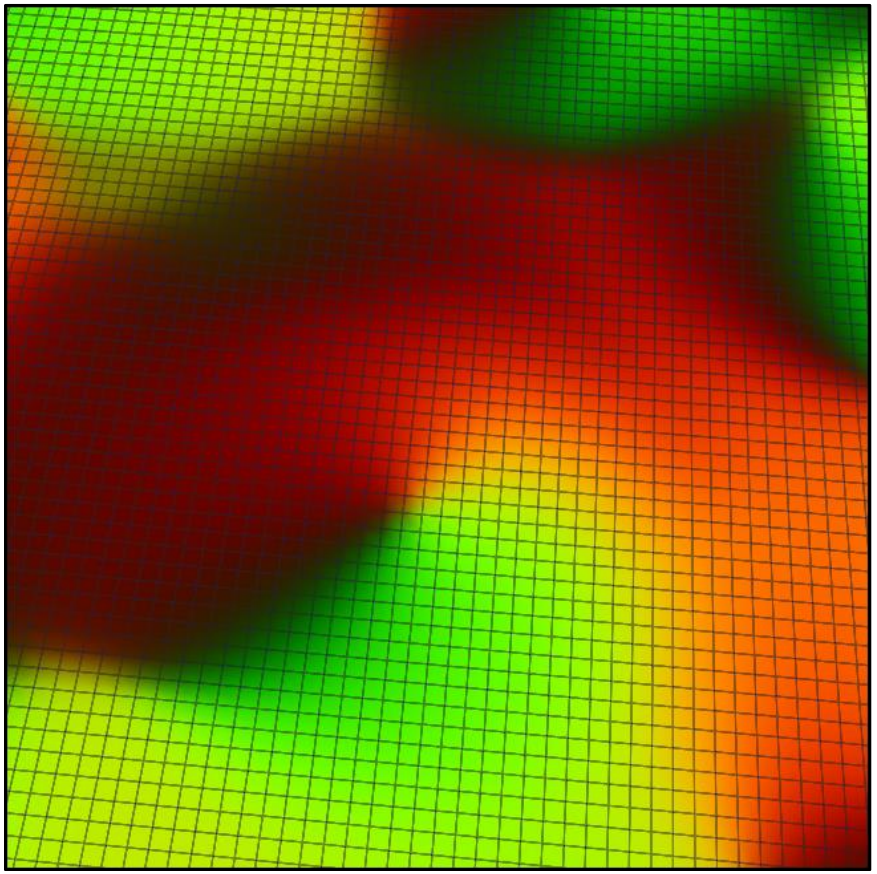
“point_scale_forces_normals” – to make normals more noticeable.



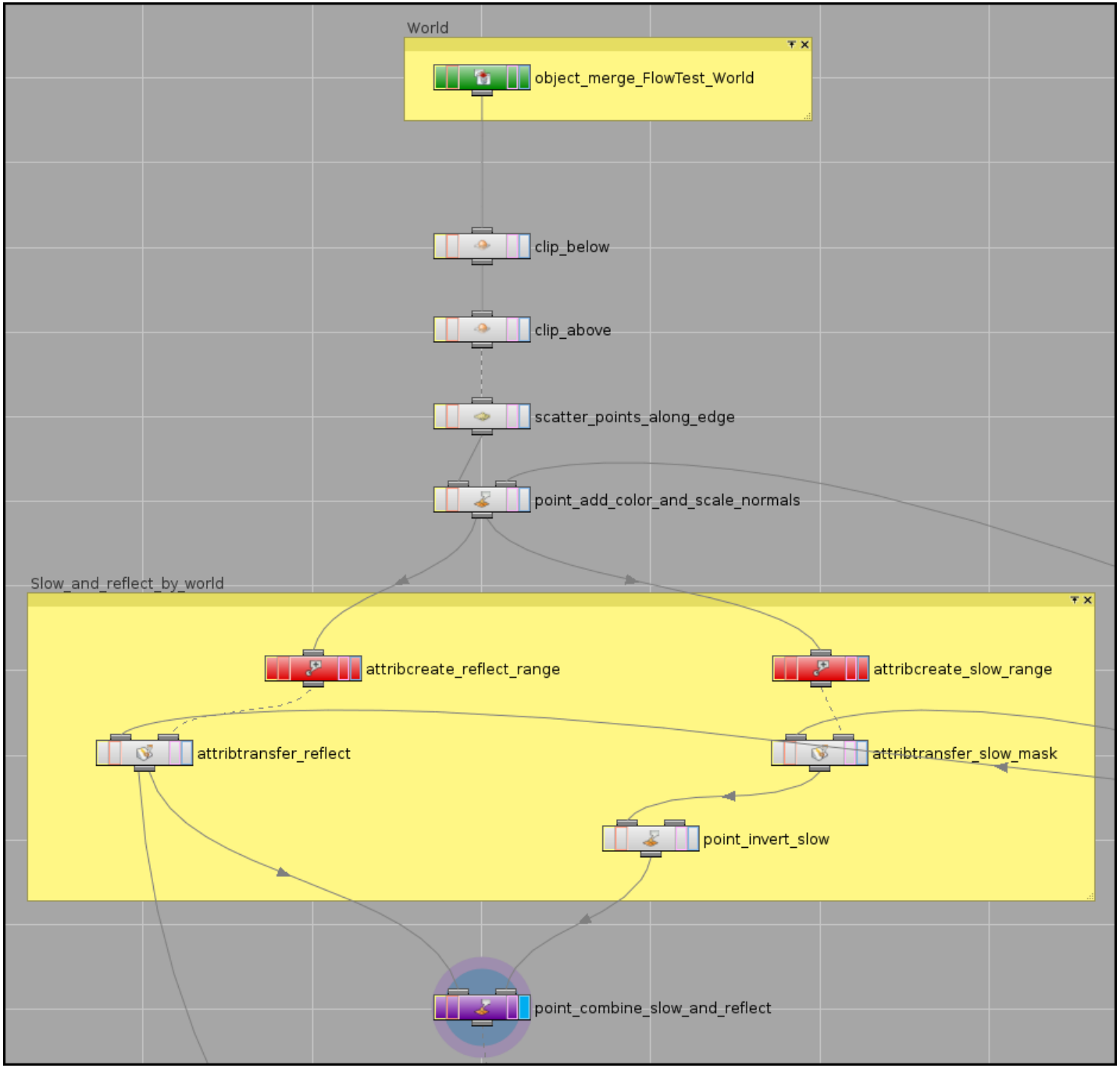
Result – turbulent surface.

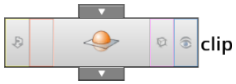


This is how it will look in color.



Step 3 – Creating reflection and slowdown of flow around world geometry



Clip 

can clip geometry according to specified plane.

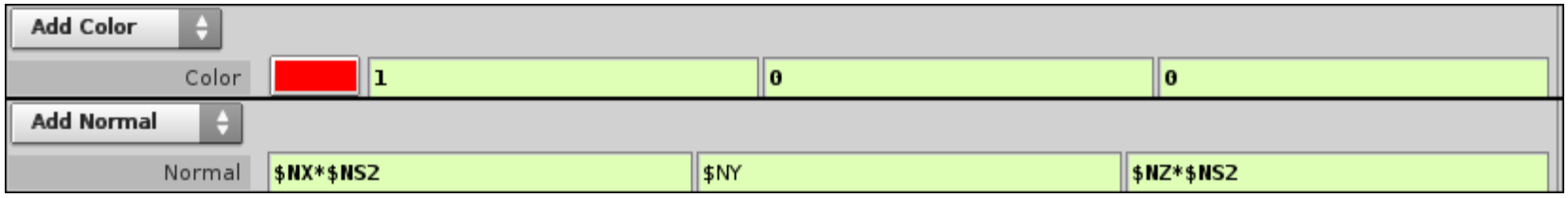
In this case, grid is parallel to XZ and it's Y = 0.
So, for "clip_below" leave all parameters by default.
"Keep" should be = Primitives Above the Plane.
It will clip geometry under water grid.

For "clip_above":
Keep = Primitives Below the Plane.
Distance = 0.01.

It will leave thin ribbons of geometry.

Then **Scatter** adds many (10000) points along this ribbons.
Points normals will be oriented outside world geometry surface, so it can be used for flow reflection.

"point_add_color_and_scale_normals" connected with "attribcreate_normals_scale" and do what expected.



I add 2 special attributes – “reflect_range” and “slow_range”. Simple floats, like “normals_scale”.

reflect_range = 150.

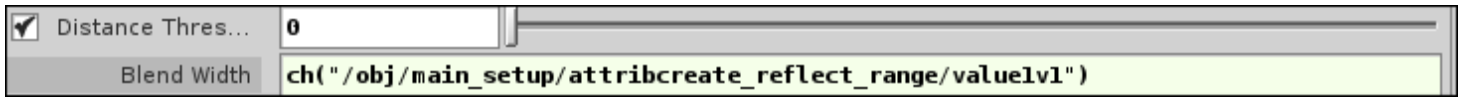
slow_range = 64.

And now, new node – AttribTransfer 

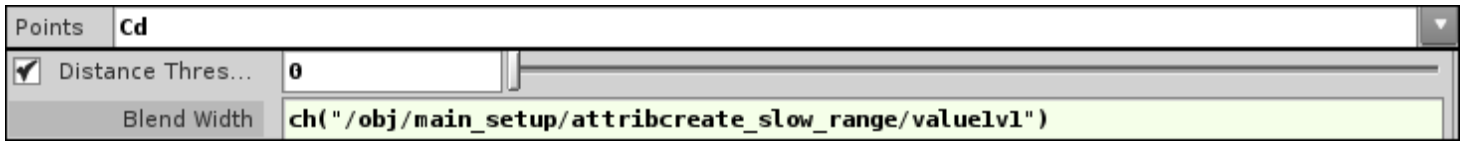
It transfers attributes from 2nd input (color and normals of points) to 1st input (grid).
Attribute can be specified if you need only 1, or not, if you need all.

In “Conditions” tab you can find “Distance Threshold” parameter. This is “hard mask” for transfer attributes.
“Blend Width” – “soft mask”.

For “attribtransfer_reflect”



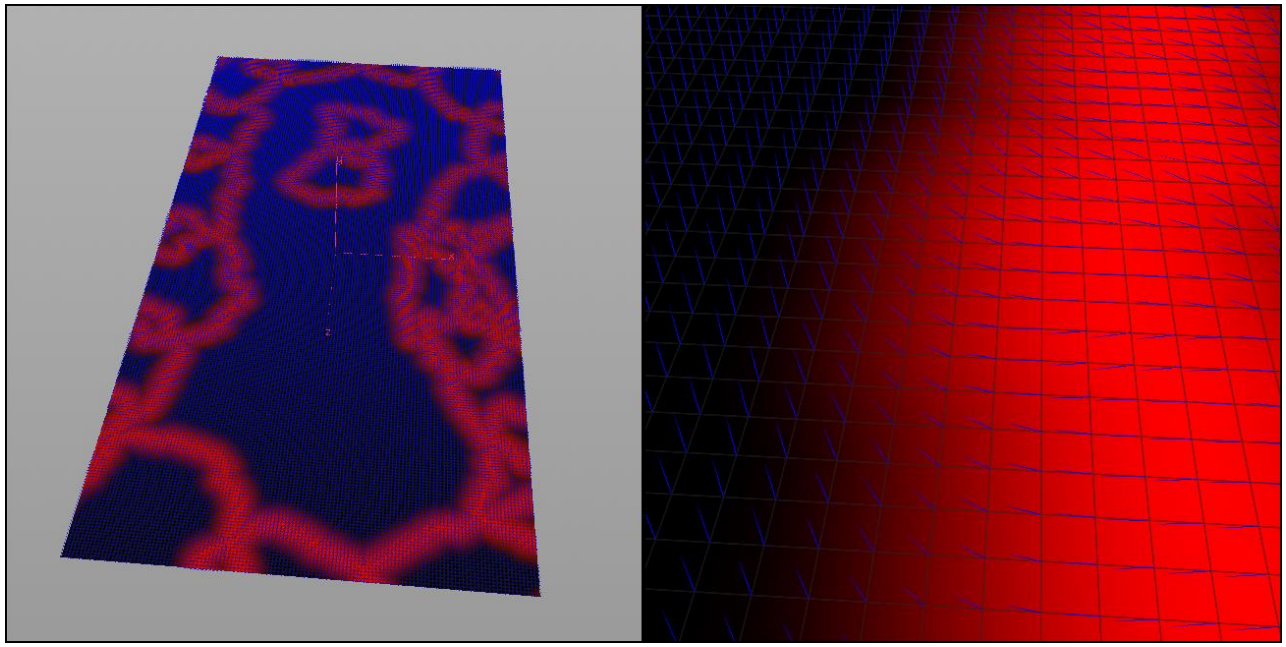
For “attribtransfer_slow”



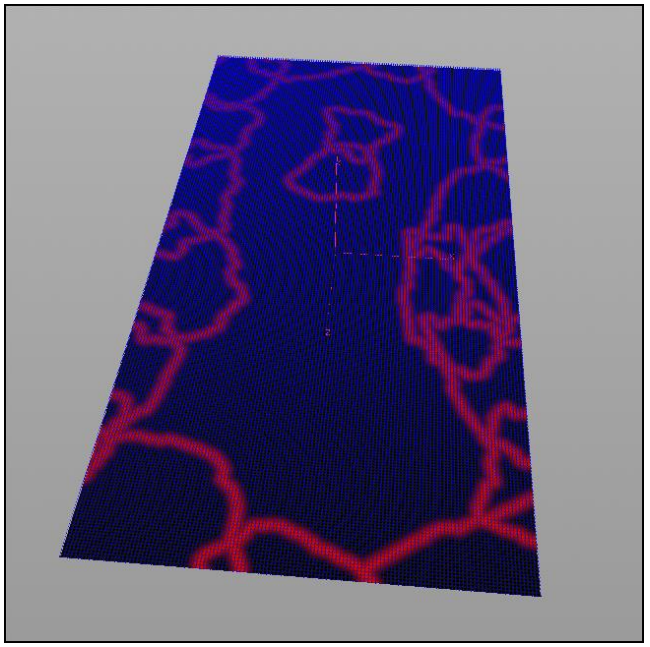
So, “range” attributes used for blend width.

Results.

Transferred color and reflected normals



Transferred slow mask



This is 3rd way for manipulating normals in **Houdini** – transferring attributes.

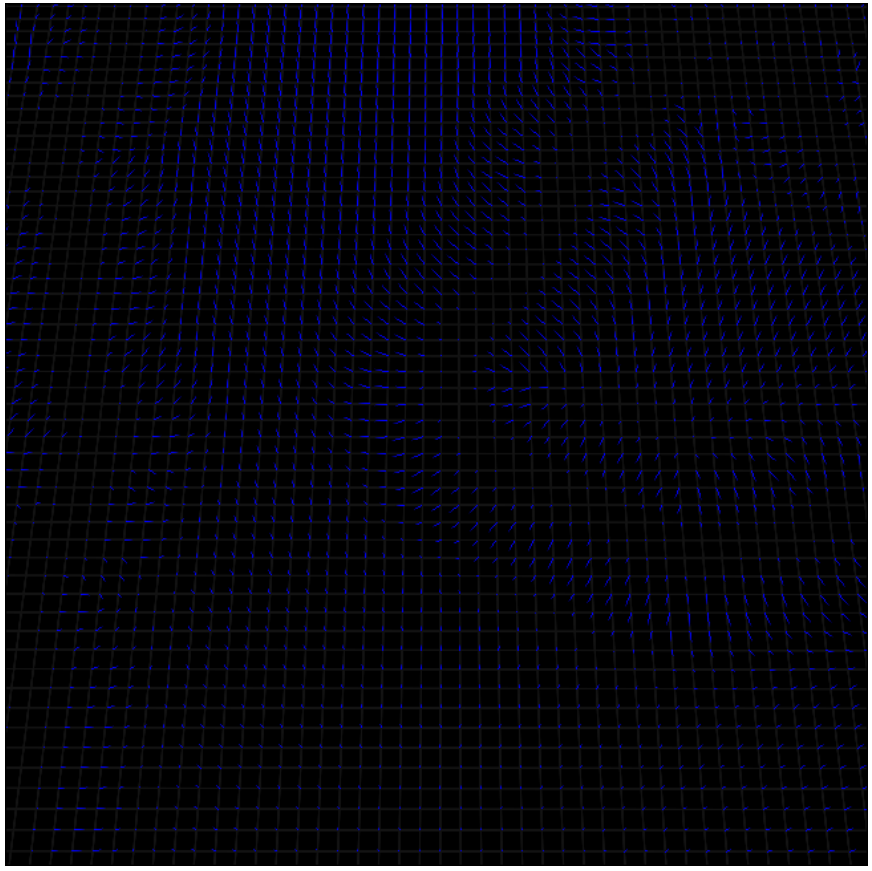
“point_invert_slow” simply inverts slow mask.

And final **Point** combines slowdown and reflection, and returns black color of grid.

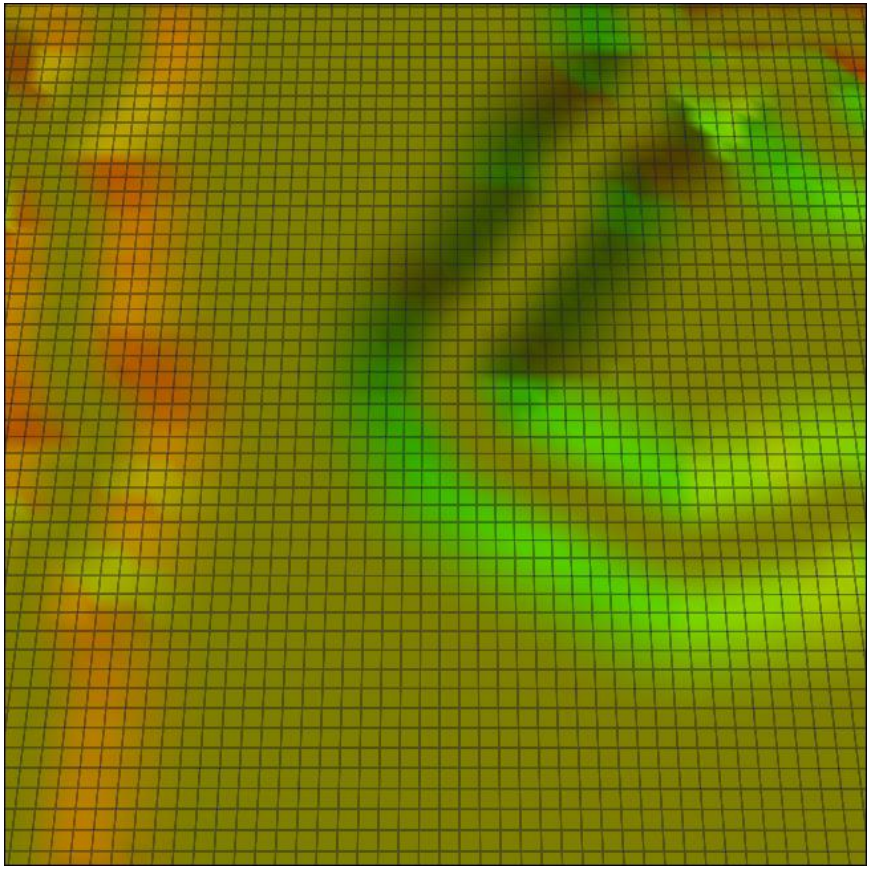
Color -> Add Color = 0, 0, 0.

Normal -> Add Normal = $NX*CR2,
$NY,
$NZ*CR2.

Result



In color



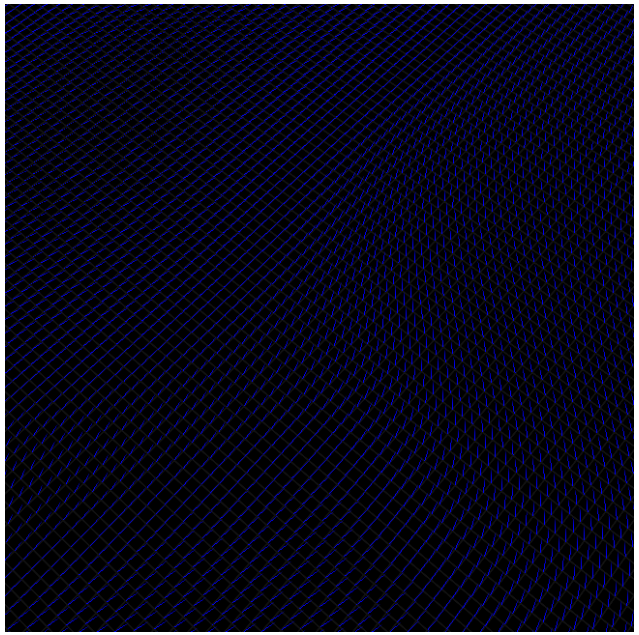
Of course reflection range can be used as mask of slow, I separate it for flexibility.
You can use different ranges for one and another.

Step 4 – Paint custom values

So here you can see already known **Comb**, and new **Paint** nodes.
Paint is the same as **Comb**, but Comb tilts normals, and **Paint** create colors on points.

Inputs:
“comb_base_flow” – default grid (“point_normals_Y_up_color_black”),
“paint_mask_slow_and_reflect” – reflection mask (“attribtransfer_reflect”)
“paint_mask_forces” – turbulent surface (“point_scale_forces_normals”)

Comb – for creating base flow.

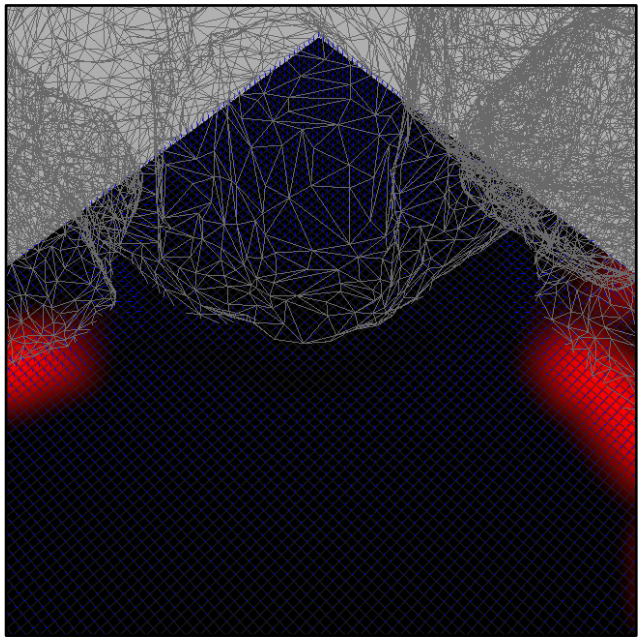


Masks for define, how much of generated effects will blend with base flow.

“paint_mask_slow_and_reflect” – for modifying slowdown and reflection mask, if you need it.
Foreground Color = 1, 0, 0.
Background Color = 0, 0, 0.

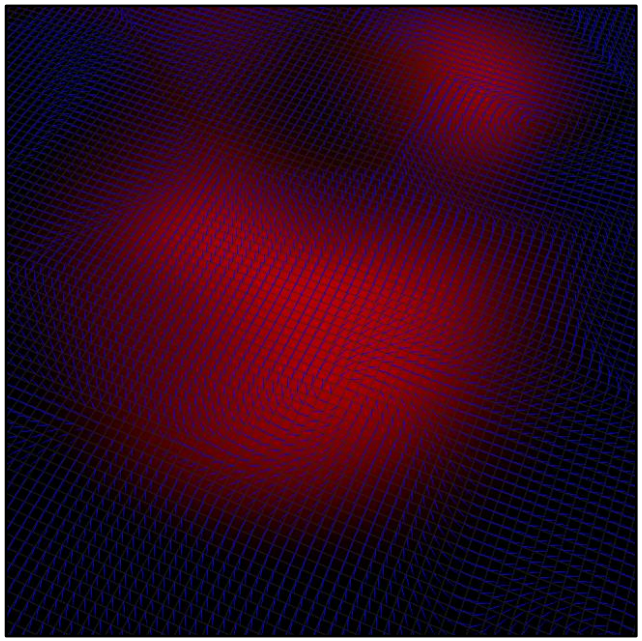
Select **Paint** node, hover **Viewport** and press Enter.
LMB for Foreground Color, MMB for Background Color.

On that level I have waterfall. I create flow from it with **Comb**,
and I don’t need any reflection and slowdown around it, so I just remove red color around it.



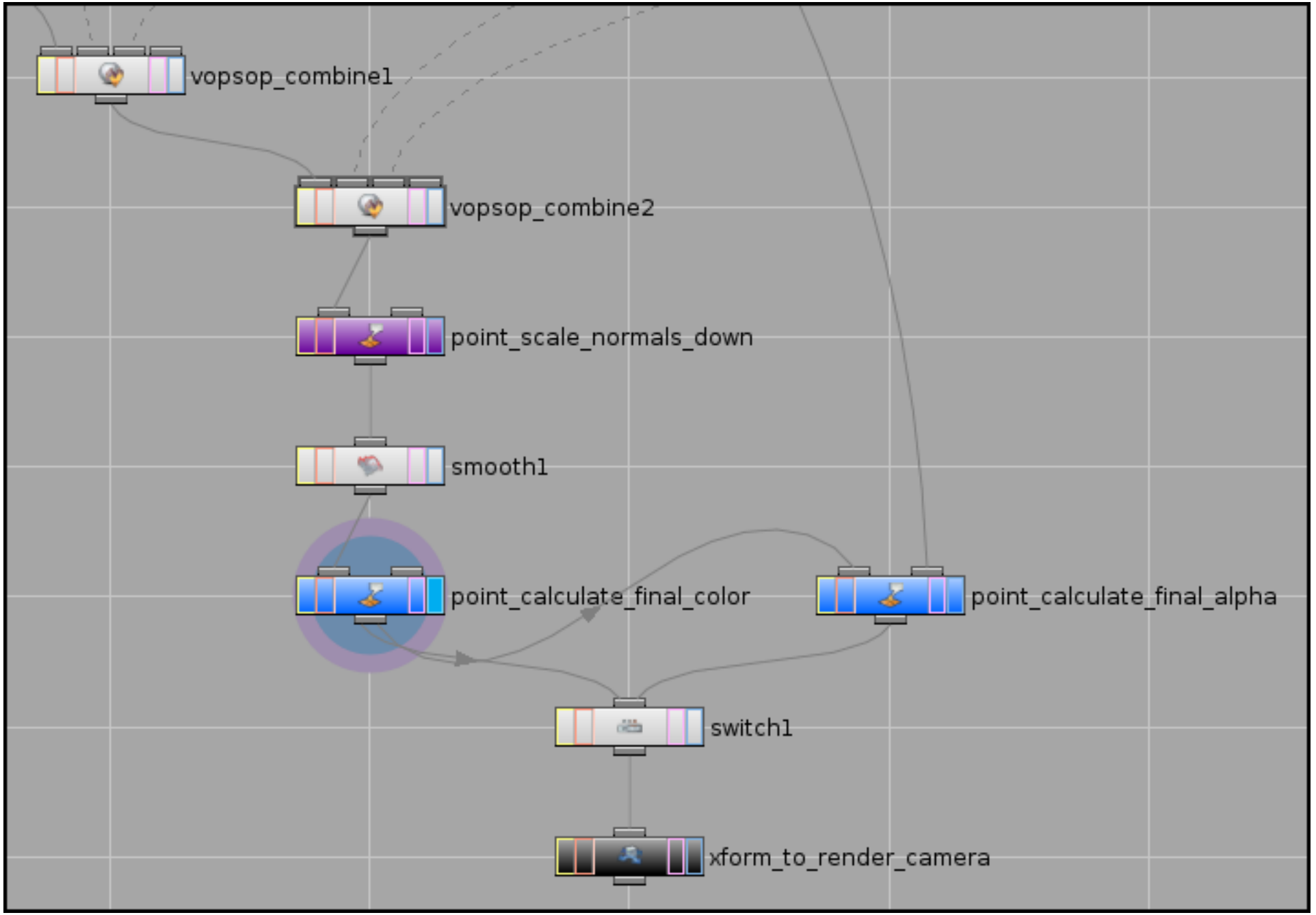
With “paint_mask_forces” I just paint few spots, where I want some upwellings and vortices.

Foreground Color = 1, 0, 0.
Background Color = 0, 0, 0.



And after this step, you already can’t change your grid tessellation.

Step 5 – Finalization



Inputs:

“vopsop_combine1” 1st – “comb_base_flow”
2nd – “point_combine_slow_and_reflect”
3rd – “paint_mask_slow_and_reflect”

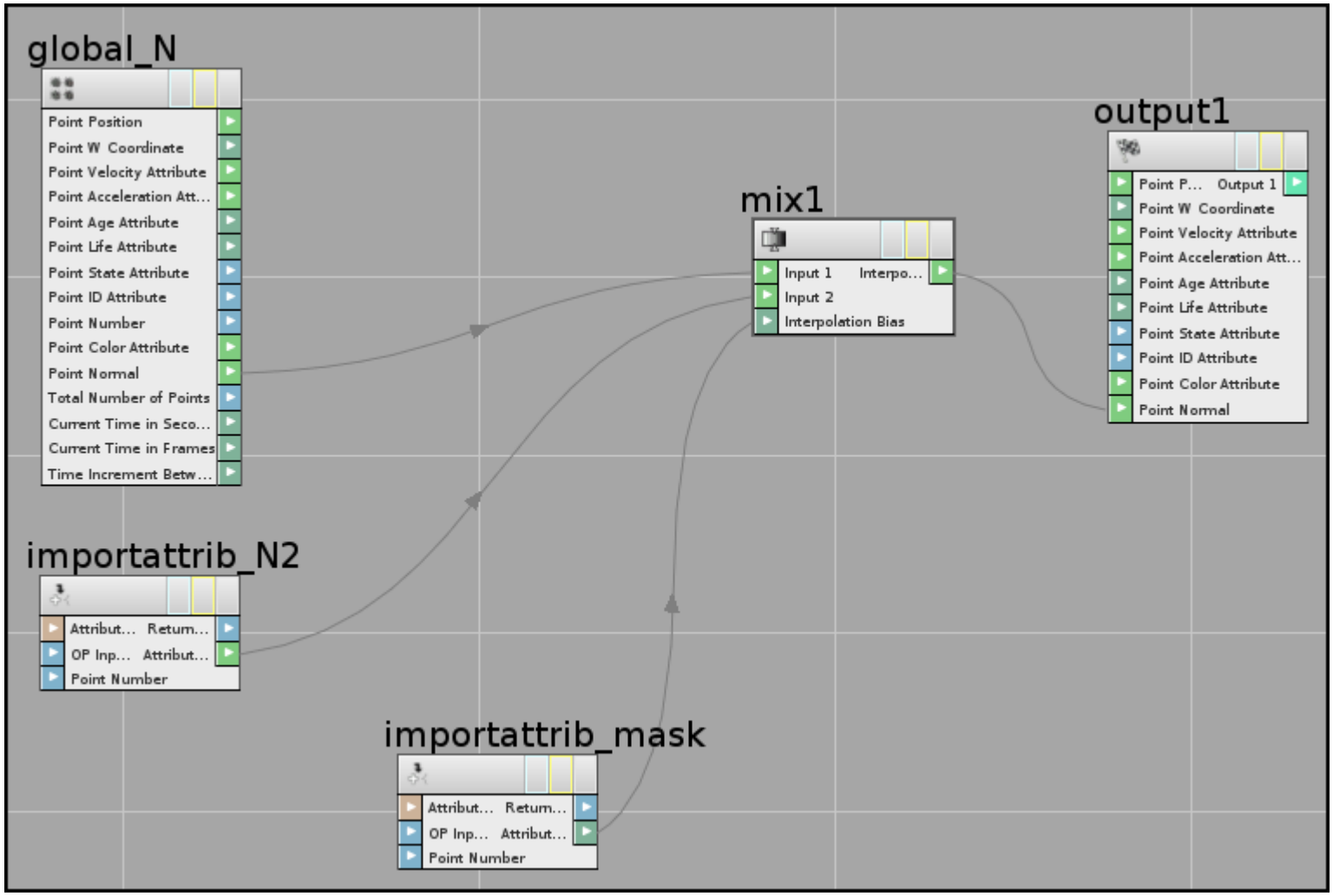
“vopsop_combine2” 1st – “vopsop_combine1”
2nd – “point_scale_forces_normals”
3rd – “paint_mask_forces”

What **VOPSOP**s doing. It gets normals from base flow and normals from effects, and using linear interpolation blends it by painted masks.

“importattrib_N2”
Signature = Vector.
Attribute = N.
OP input index = 1.

“importattrib_mask”
Signature = Float.
Attribute = Cd.
OP input index = 2.

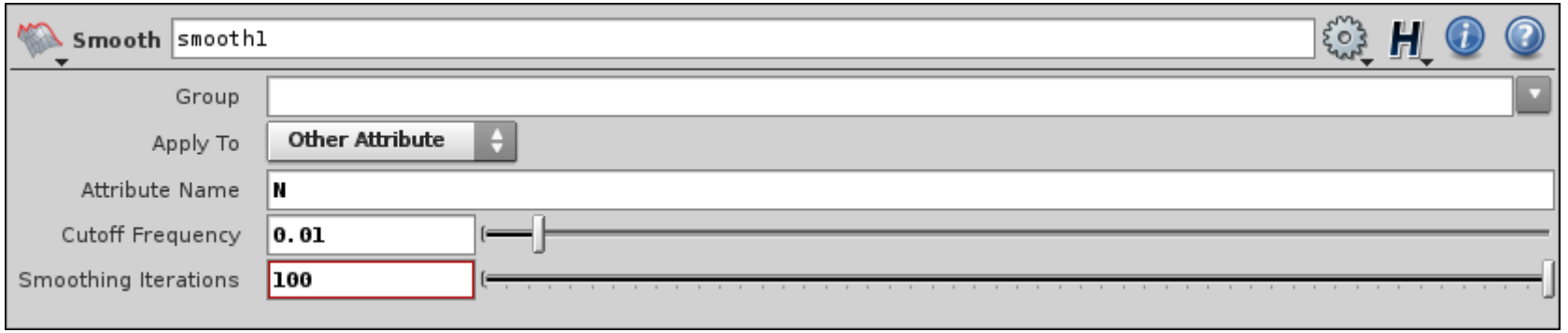
“mix1”
Signature = 3D Normal.



“point_scale_normals_down”
Normal -> Add Normal = $NX/$NS,
$NY/$NS,
$NZ/$NS.

Smooth 

simply smooth a bit final normals.

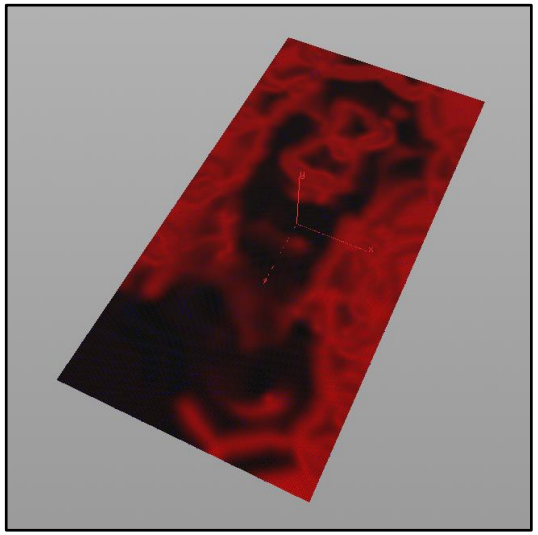


“point_calculate_final_color” you already know.
Normal -> Add Normal = ($NX*0.5 + 0.5),
($NZ*0.5 + 0.5),
0.

“point_calculate_final_alpha” calculates alpha mask for foam texture.

So, the smaller flow speed, the more foam accumulated, and vise versa.
Color -> Add Color = 1 - length($NX, 0, $NZ),
0,
0.

In shader, I add 2 additional textures with heavy foamed water, which interpolates with simple water texture, and have same flow distortion.



In this case I have that waterfall, and I want to get many of foam around it.
So, I simply repeat all steps which I used for slowdown, with waterfall mesh.

It intersect XZ plane, a bit, and everything works fine. If not, simply tweak **Clip**'s "Origin" Y parameter and use larger "Blend Width" value of "attribtransfer_waterfall_foam", in any case, attributes will be transferred.

1st input of "attribtransfer_waterfall_foam" is default grid.
Points = Cd.
Distance Threshold = 0.
Blend Width = 500.

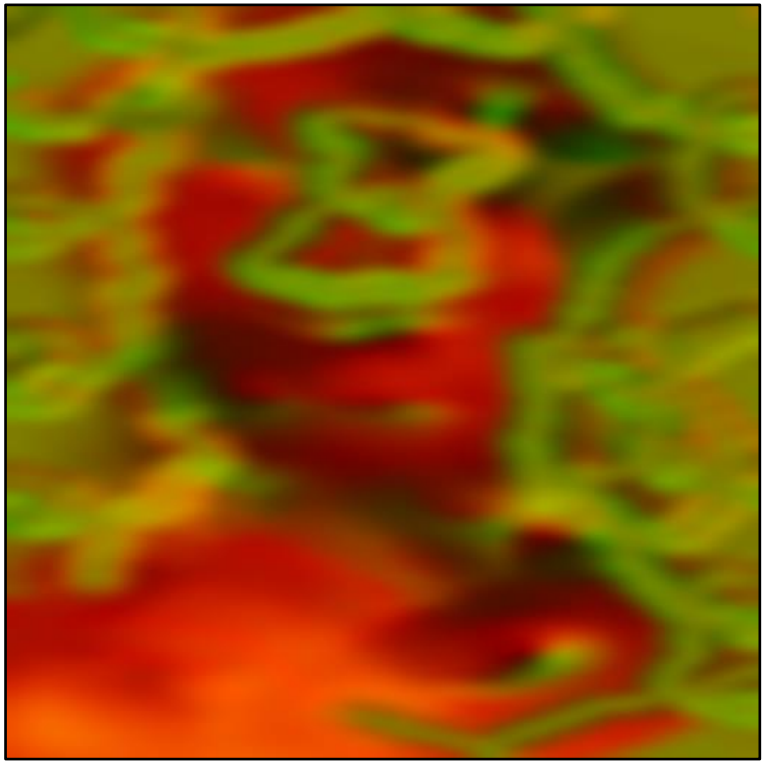
And now, connect “attribtransfer_waterfall_foam” to 2nd input of “point_calculate_final_alpha”.

Color -> Add Color = 1 - length($NX, 0, $NZ) + $CR2,
0,
0.

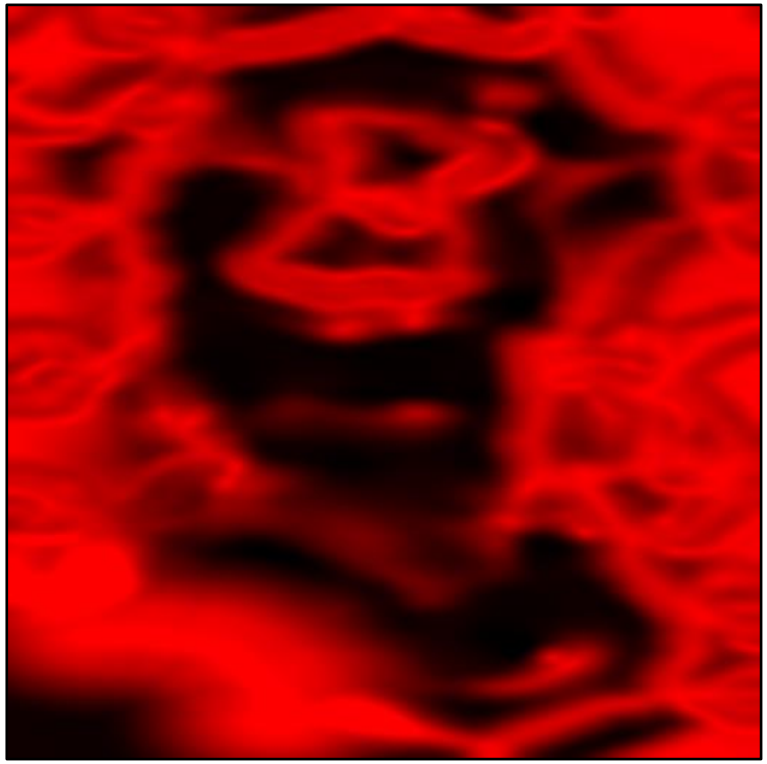
As result – big spot of waterfall foam on it’s place.

Finally, **Transform** to render camera, with
Translate = 0, 0, -1024.
Rotate = 90, 0, 0.
Scale = 1, 1, 0.5.

Flow map

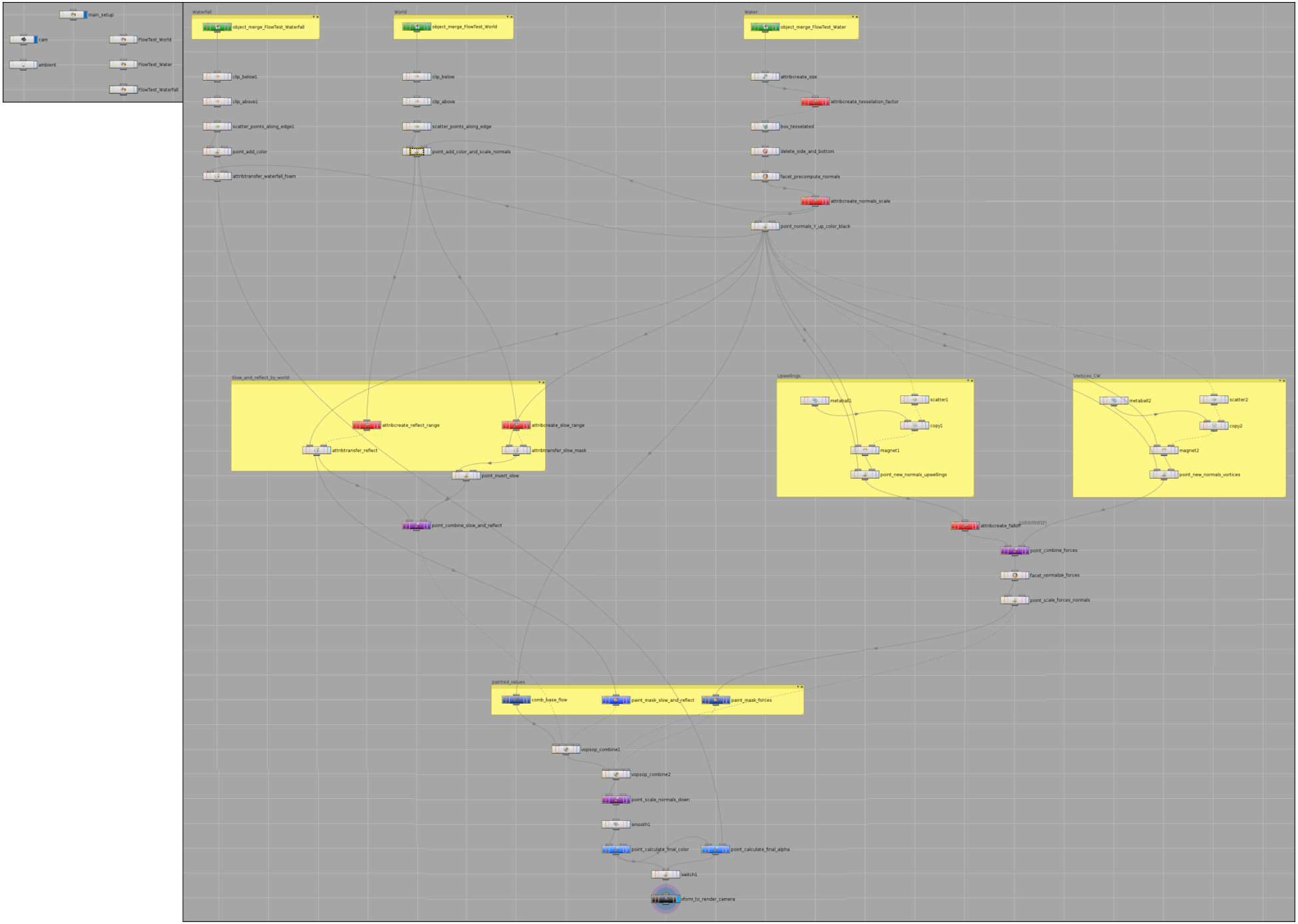


Foam mask



[Video with final map on YouTube.]

Final network.



Textures creation

Also, I want to explain how normal maps and noise map for this tutorial was created.

Normal map

There is one cool tool for Houdini – **Houdini Ocean Toolkit** (**HOT**).
Old repository – http://code.google.com/p/houdini-ocean-toolkit/
New repository – https://github.com/eloop/hot/
And some build for Houdini 12 – http://schnellhammer.net/blog/2012/03/hot12/

I use one from last link.

HOT provides functionality for creating water waves on some tessellated grid.

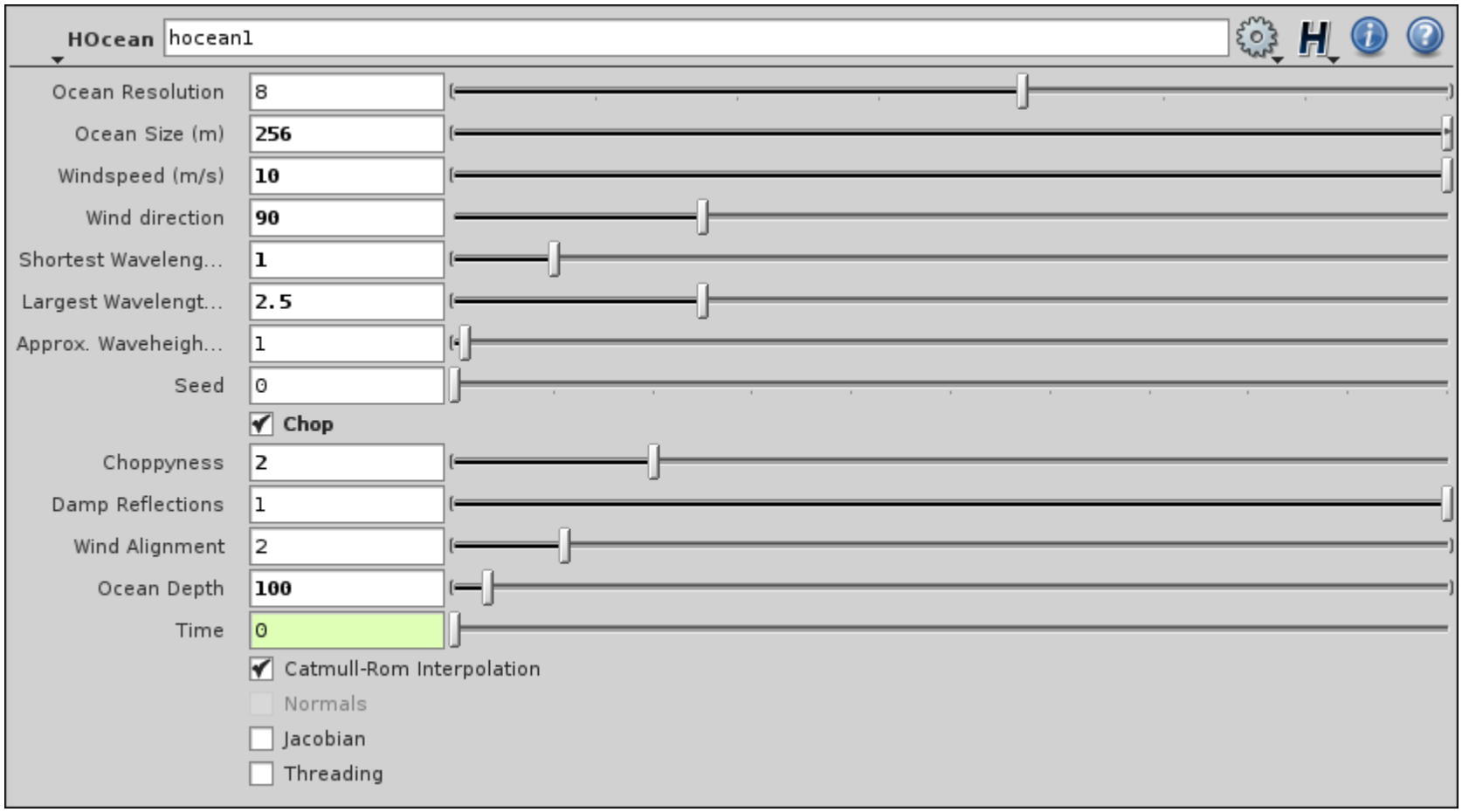
So, install **HOT**, open Houdini, create **Grid**, **Camera** and **Ambient Light**.

Camera:
Translation = 0, 128, 0.
Rotate = -90, 0, 0.
Icon Scale = 100.
Resolution = 1024*1024.
Aperture = 100.

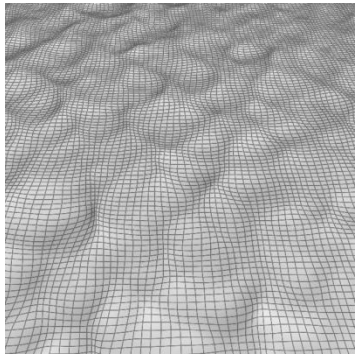
Ambient Light:
Ambient color = 1, 1, 1.
Light Intensity = 1.

Grid:
Size = 512*512.
Rows/Columns = 512*512.

TAB -> **HOcean.**

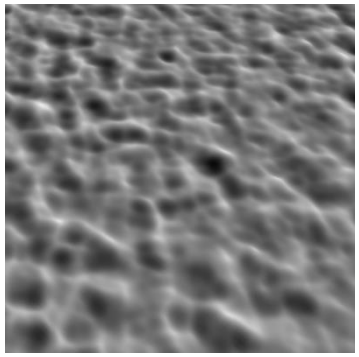


Now you have some waves:

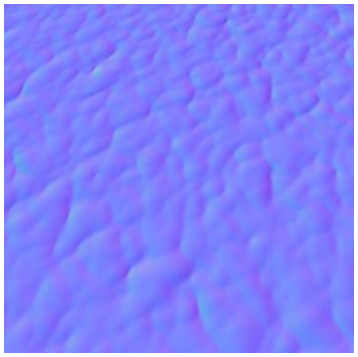


TAB -> **Point**, 2 times.
Maybe you want get height map of this waves, maybe normal map. In any case, it not tileable. It almost, but not...

“point_height”
Color -> Add Color = fit($TY, $YMIN, $YMAX, 0, 1),
fit($TY, $YMIN, $YMAX, 0, 1),
fit($TY, $YMIN, $YMAX, 0, 1).



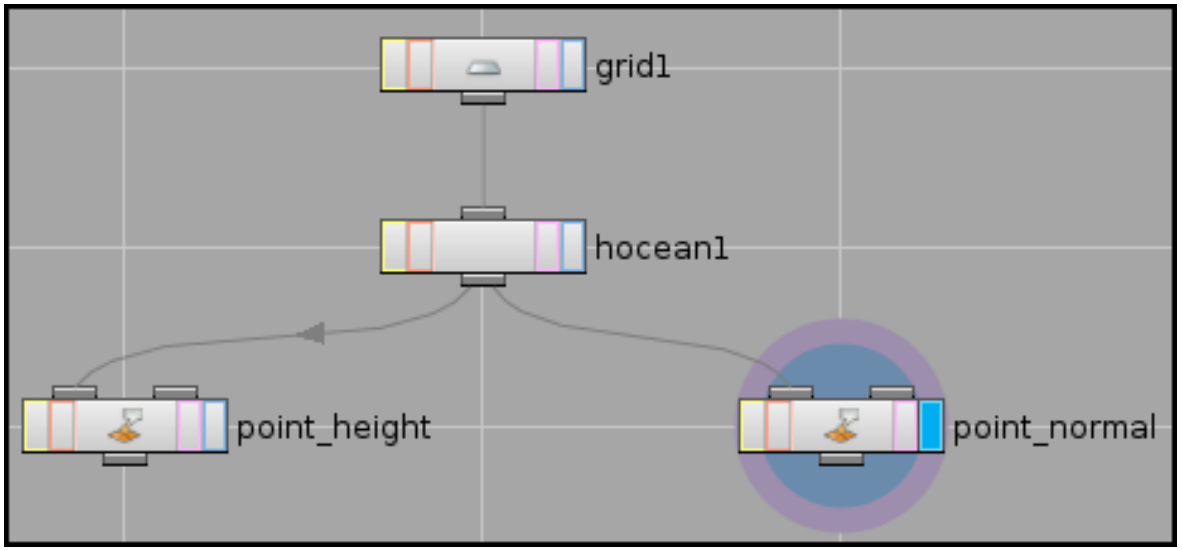
“point_normal”
Color -> Add Color = $NX*0.5+0.5,
-$NZ*0.5+0.5,
$NY.



Now, go to **Render View** and get what you need.

I get 2 samples of height map using “Seed” parameter of **HOcean**, and then manually scale it, merge, make tileable and generate normal maps.

Final **Network**

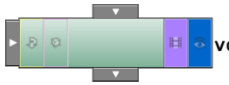


Noise map

Go to **Composite View**.

Network View must switch and you will find **Image Network** 

Open it.

TAB -> **VOP COP2 Generator** 

Open it.

Now you can see **Global** and **Output** nodes.

TAB -> **Float To Vector**.

Connect “Pixel Horizontal Position” and “Pixel Vertical Position” with **Float To Vector** 1st and 2nd components.

TAB -> **Unified Noise**.
Frequency = 20, 20, 1, 1.
Period = 20, 20, 1, 1.
Noise Type = Periodic Perlin.

TAB -> **Vector To Float**.

Connect **Vector To Float** 1st/2nd/3rd components with “Pixel Red/Green/Blue Value” of **Output**.

Go to Main Menu -> Edit -> Compositing Settings, and set Resolution to 512*512.

Flow Map import in UDK

Flow maps should be imported as normal maps, to be unpacked from -1 to 1.

For this tutorial used “NormalMapUncompressed” for better distortion quality.

Conclusion

As you can read in **Naughty Dog** docs, it's useful for lots of other effects – clouds, sand, snow, etc.

Special thanks to:

Nikolas
Green-Man
edward
Niana

for help in making this tutorial.

That’s all.
Good luck!

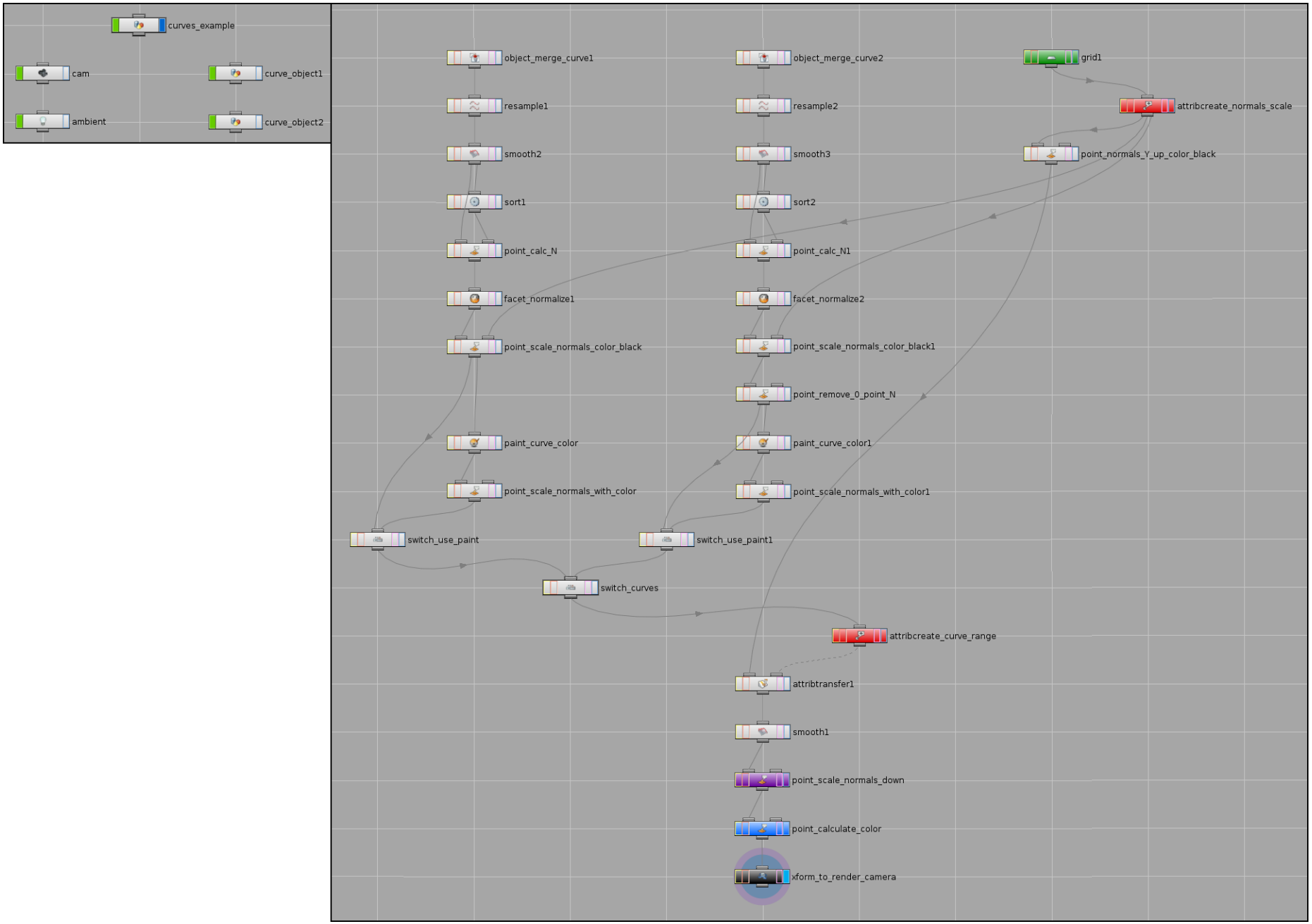
Downloads

This document

Archive with all stuff

Appendix A – Creating flow with curves

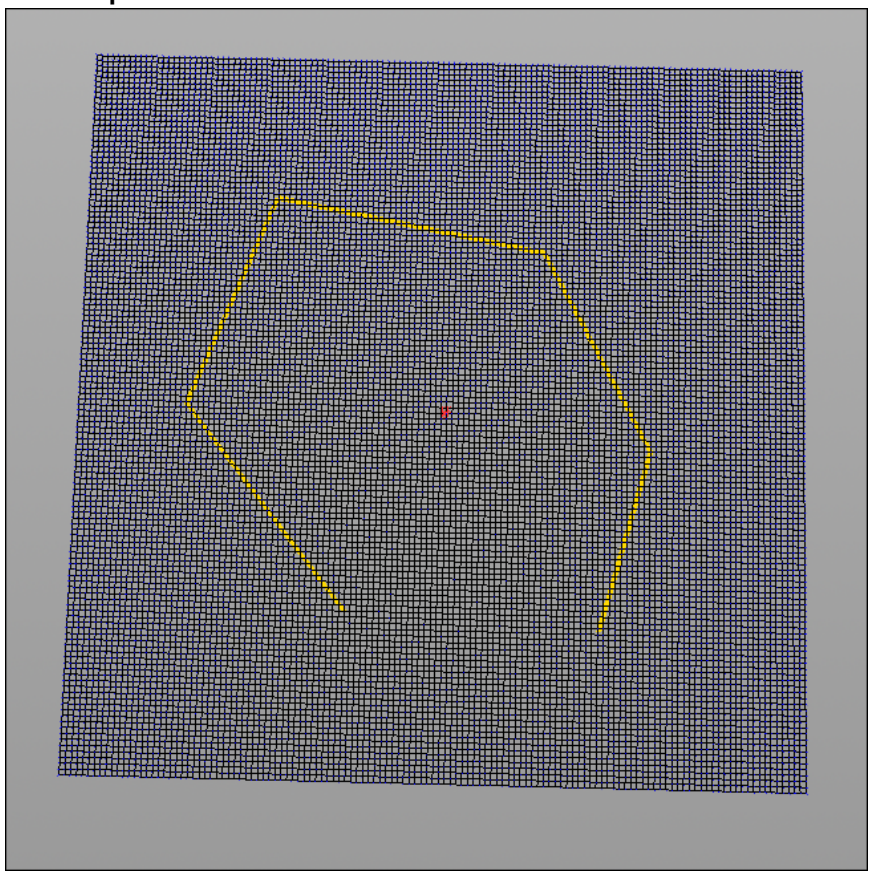
As alternative for **Comb** tool, you can use simple polygonal curves to create base flow.

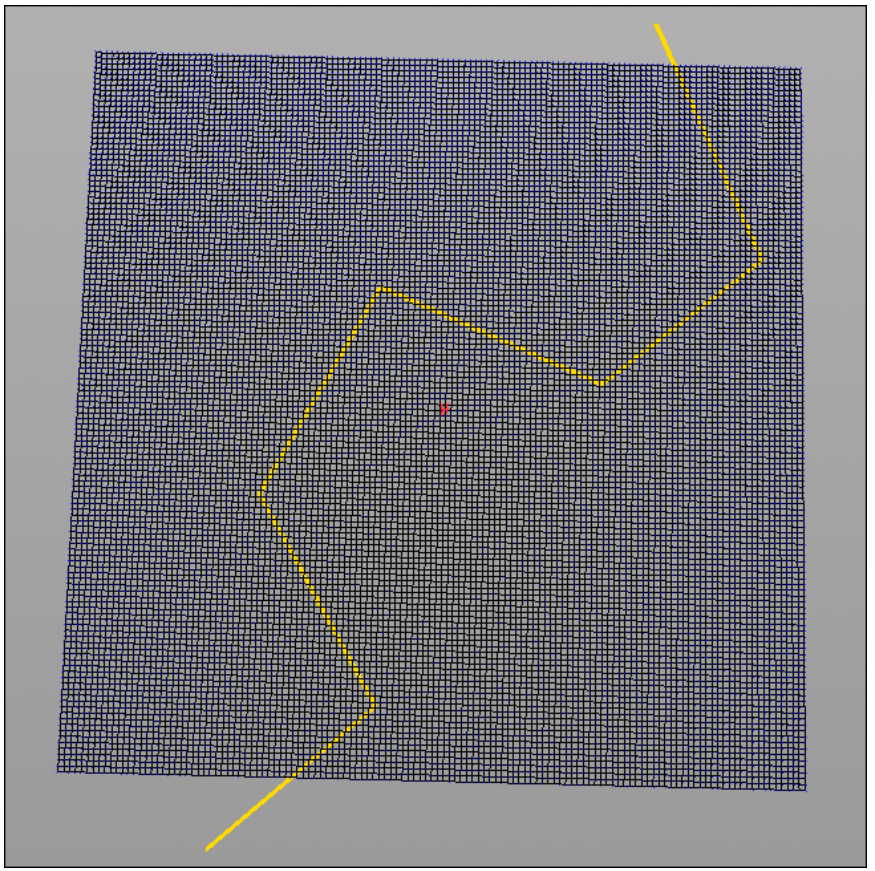


Create **Grid**, **Camera** and **Ambient Light**.

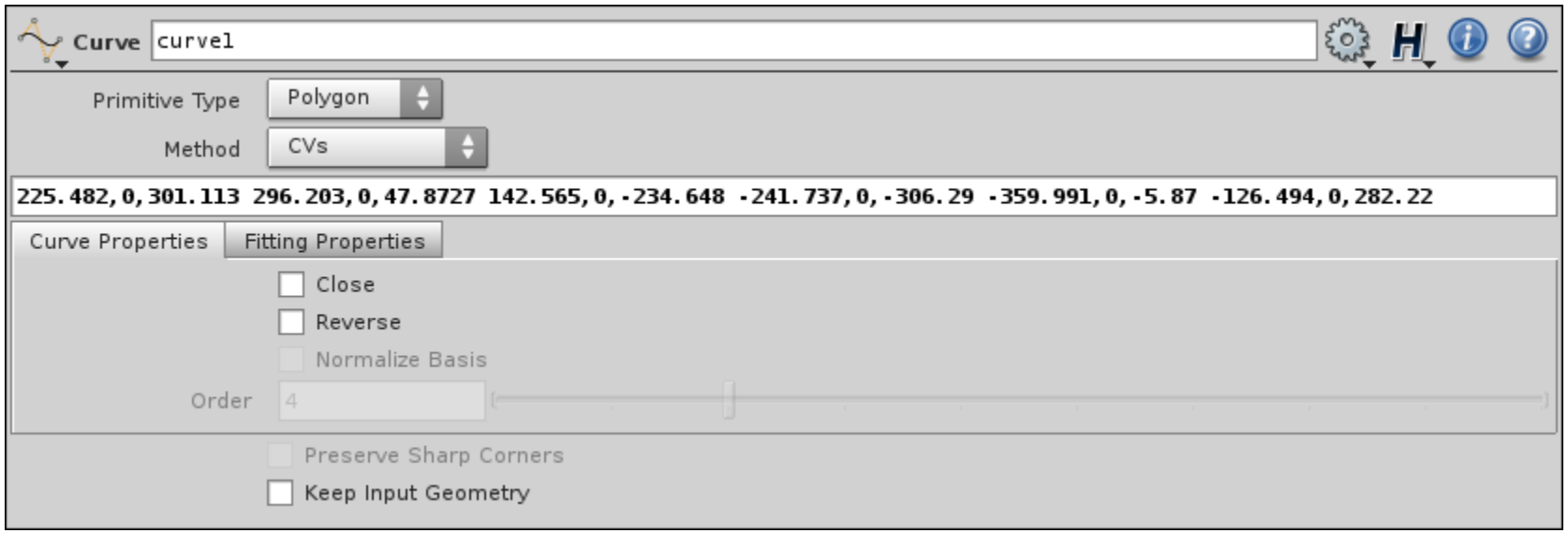
On **Create Shelf** select **Curve** 
In **Viewport**, hover cursor on grid and place few points with LMB. Press Esc to finish operation.
Create 2 different curves.

Here is preview in “Wireframe” mode:





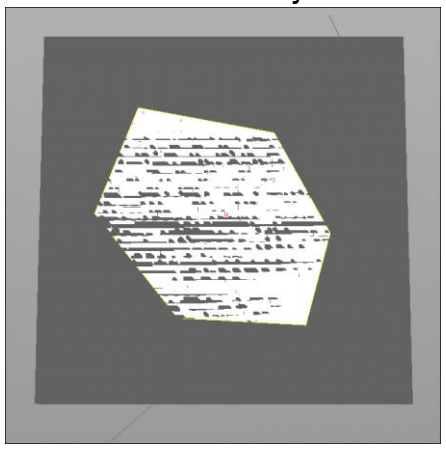
Open one in **Network**, select internal curve object and look on it's properties.



Here you can see:
Coordinates of every point (225.482,0,301.113 296.203,0,47.8727 ...),
Primitive Type = Polygon.
"Close" checkbox.

So, curves should be polygonal and can be closed or not.

Close one of it.
Houdini automatically connect first and last points of curve, and add polygons inside it.



Return to **obj** network and open grid object.
Add “attribcreate_normals_scale” and “point_normals_Y_up_color_black”.

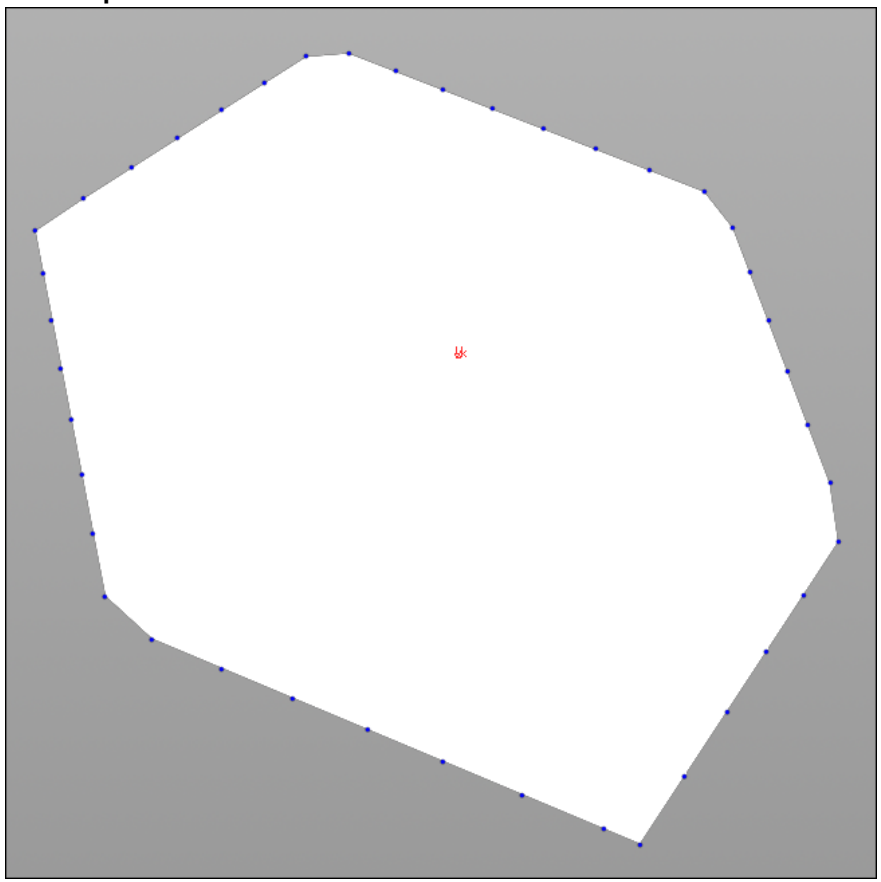
Now merge both curves with this network using **Object Merge**.
For now curves are very angular, and it needs to be resampled and smoothed.

TAB -> **Resample** 

This operator adds points to curve.
Connect **Resample** with **Object Merge** of closed curve.

“Length” parameter means 1 point per distance. Set Length = 50.

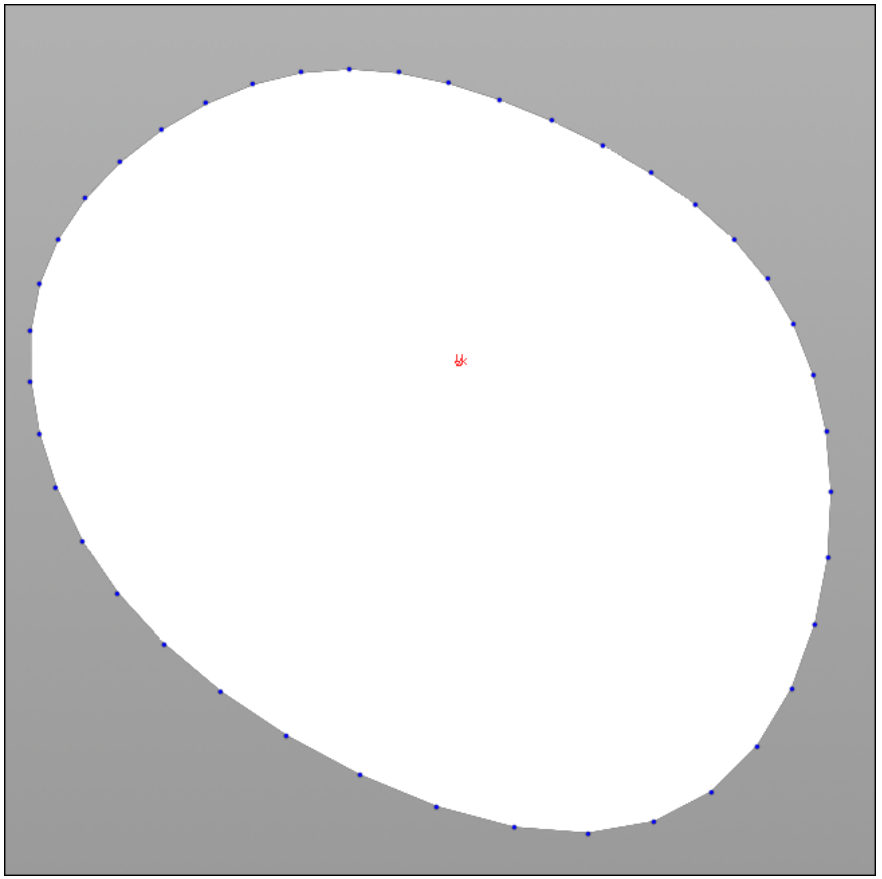
Resample.



TAB -> **Smooth**, to smooth curve.

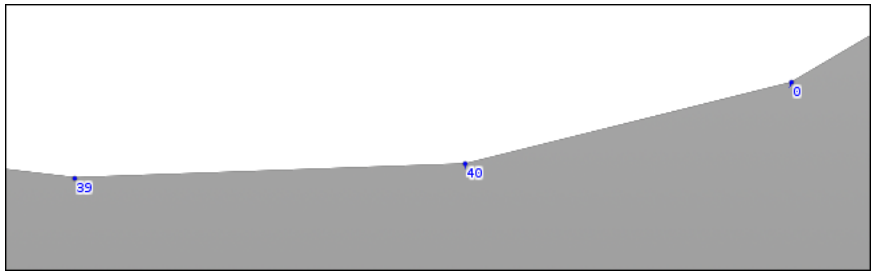
Cutoff Frequency = 0.0765.
Smoothing Iterations = 100.

Smooth.

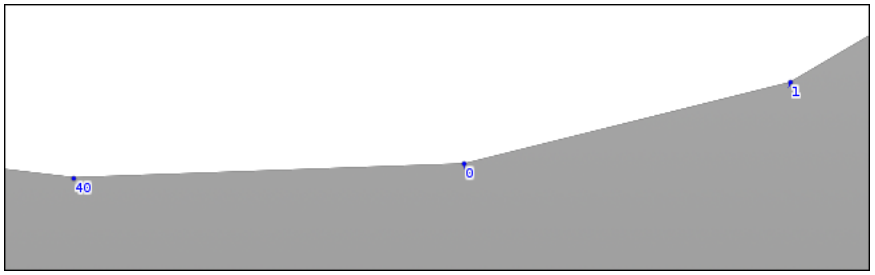


Normals need to be oriented along curve.
TAB -> **Sort**
Point Sort = Shift.
Offset = 1.
It will shift points order by 1.

Old order.

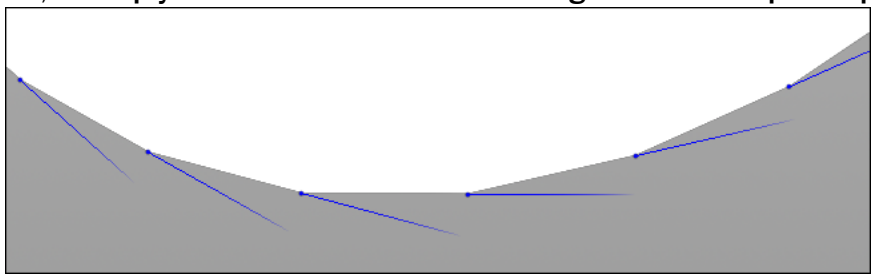


New order.

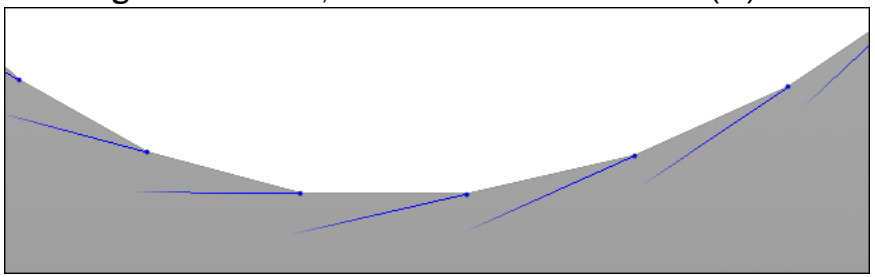


TAB -> **Point**
Normal -> Add Normal = $TX-$TX2, 0, $TZ-$TZ2.

So, it simply calculates new normals using old and new points positions.

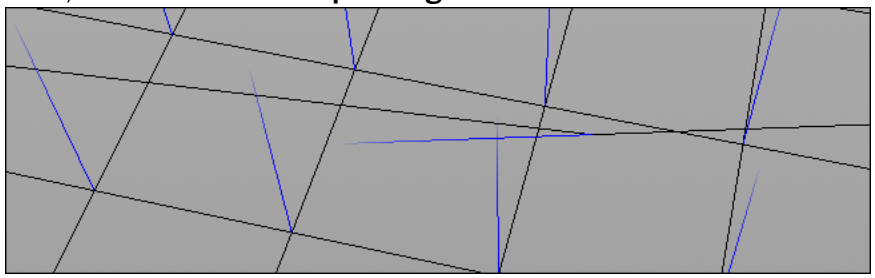


To change it's direction, in **Sort** node set "Offset" to (-1).

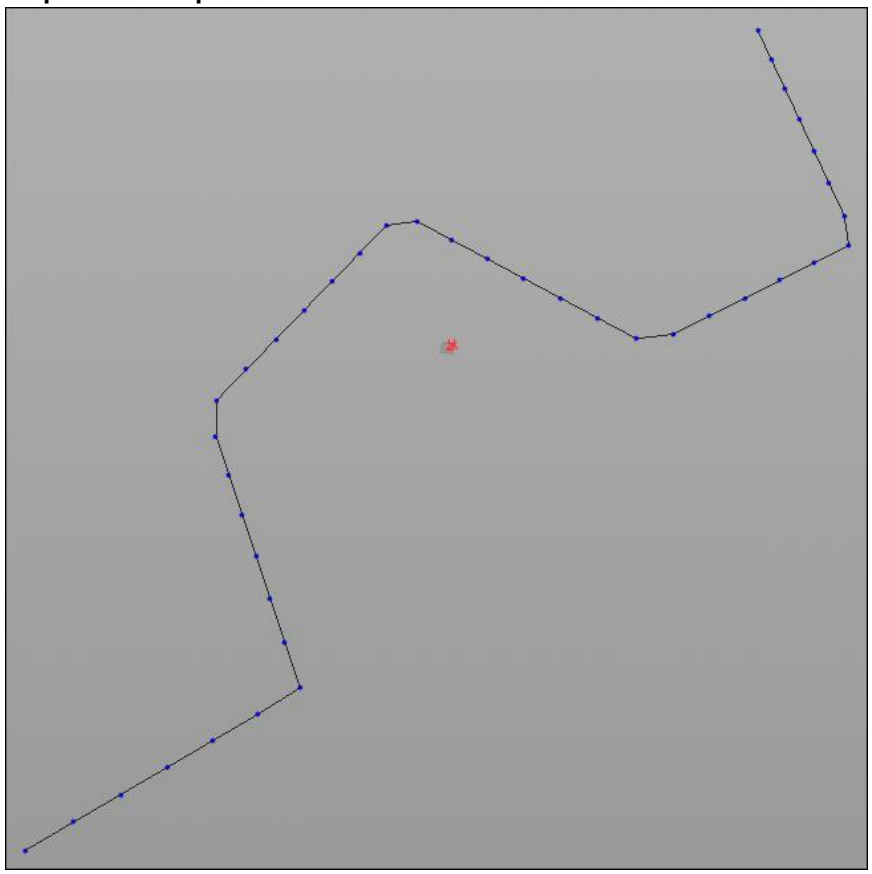


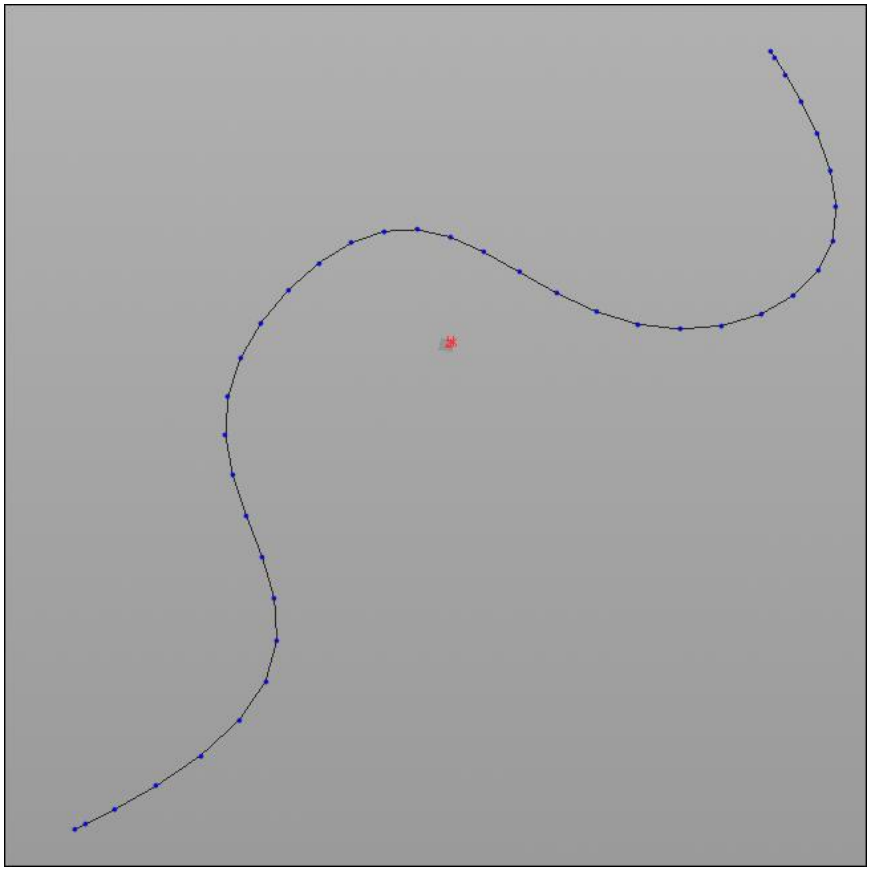
TAB -> **Facet**, "Make Normals Unit Length" for normalizing.
TAB -> **Point**, for scale normals to NS and set color to black.

Now, curve's normals equal to grid's normals.

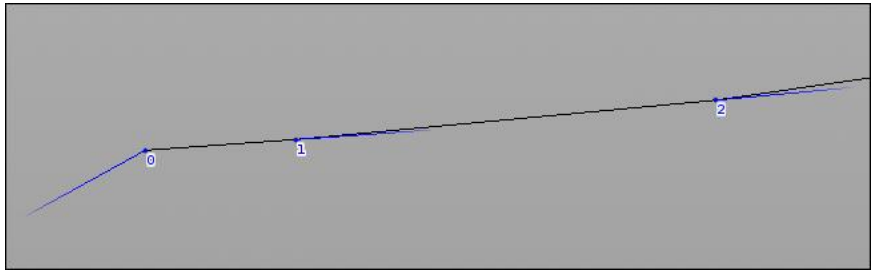


Repeat all steps for second curve.



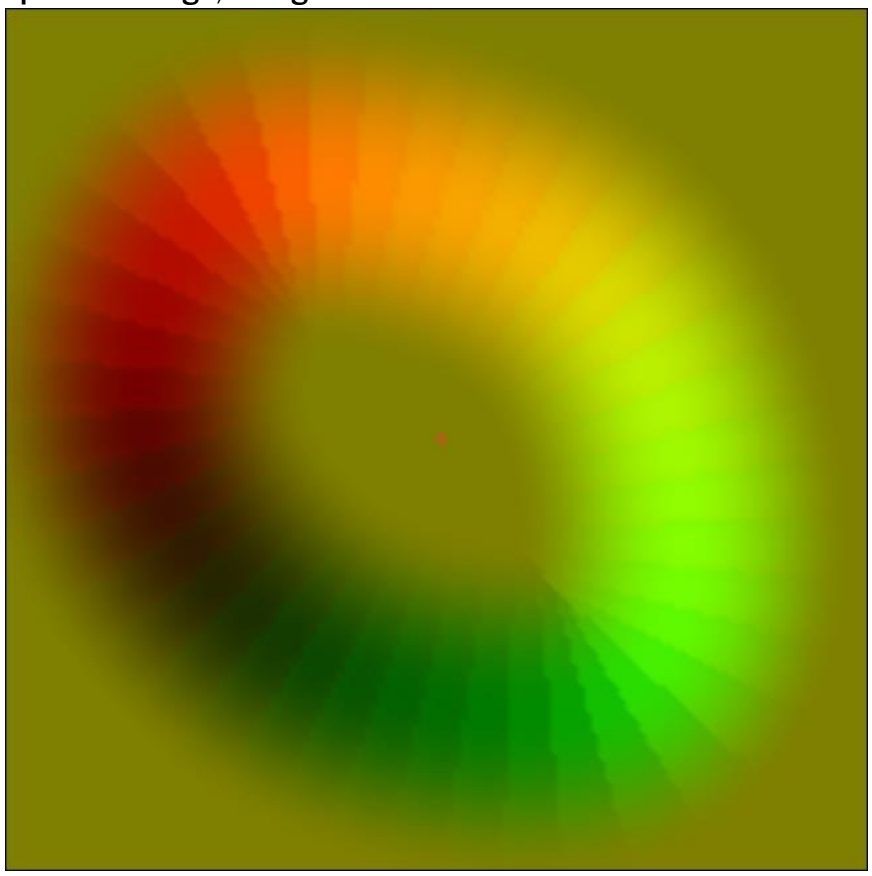


Because this curve are opened, after sorting and calculations, it's 0 point normal becomes bad.

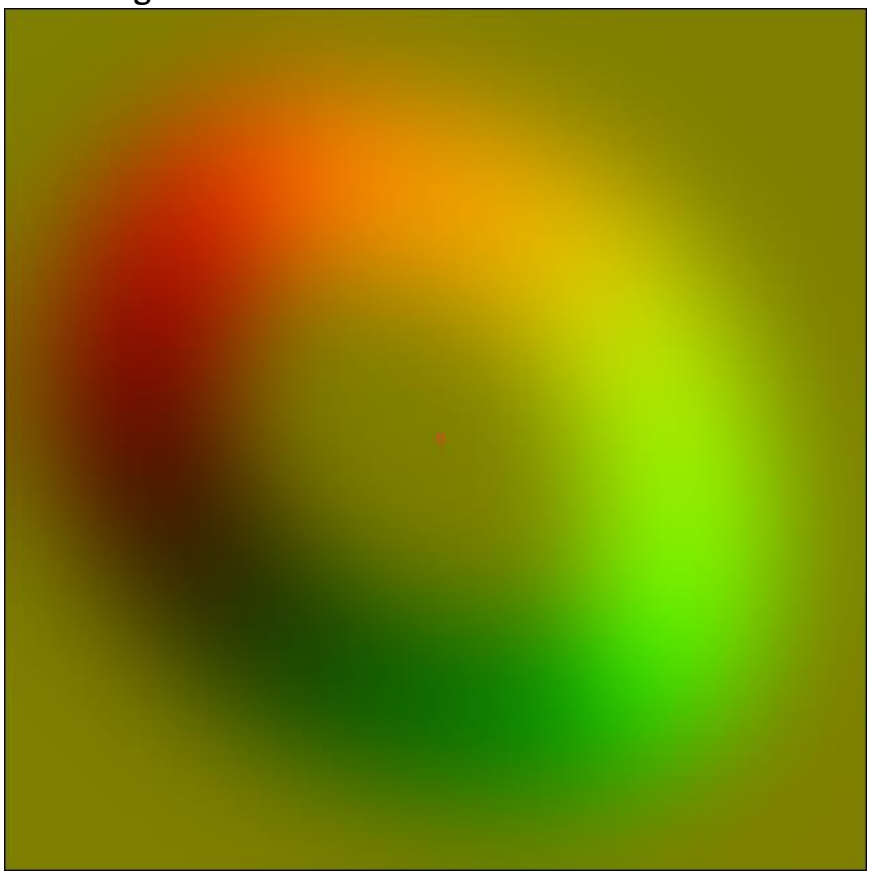


Simply remove it with
TAB -> **Point**
Group = 0.
Normal -> Add Normal = 0, 0, 0.

For now, you can transfer curve's normals to grid, with specified range, and get some flow.

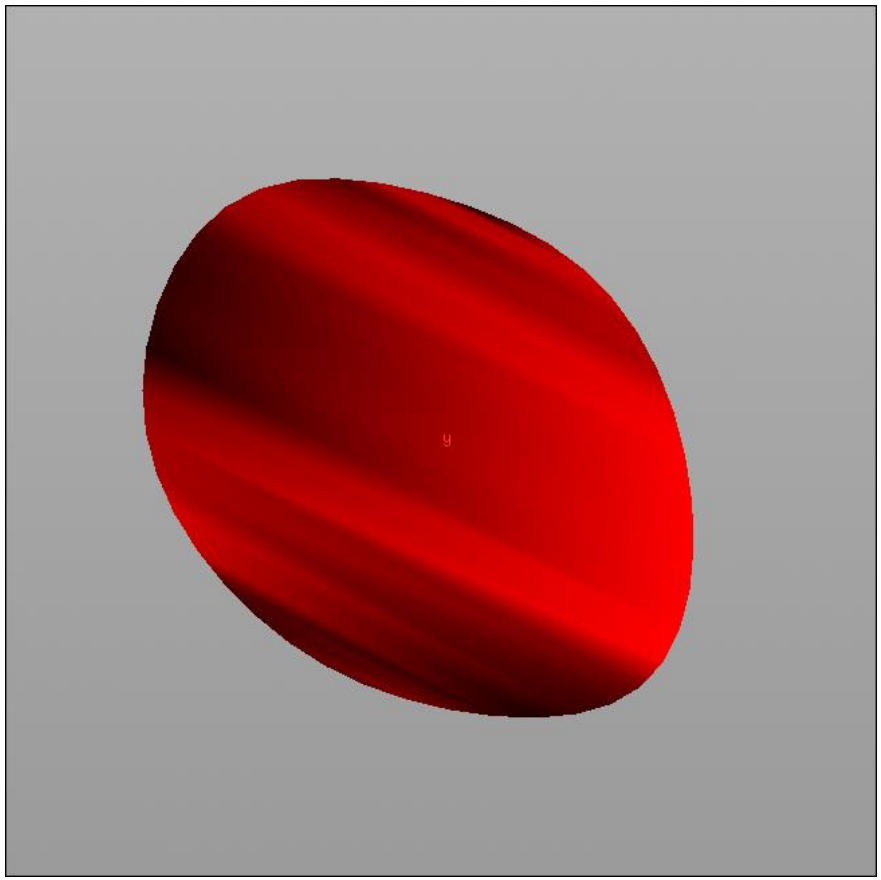


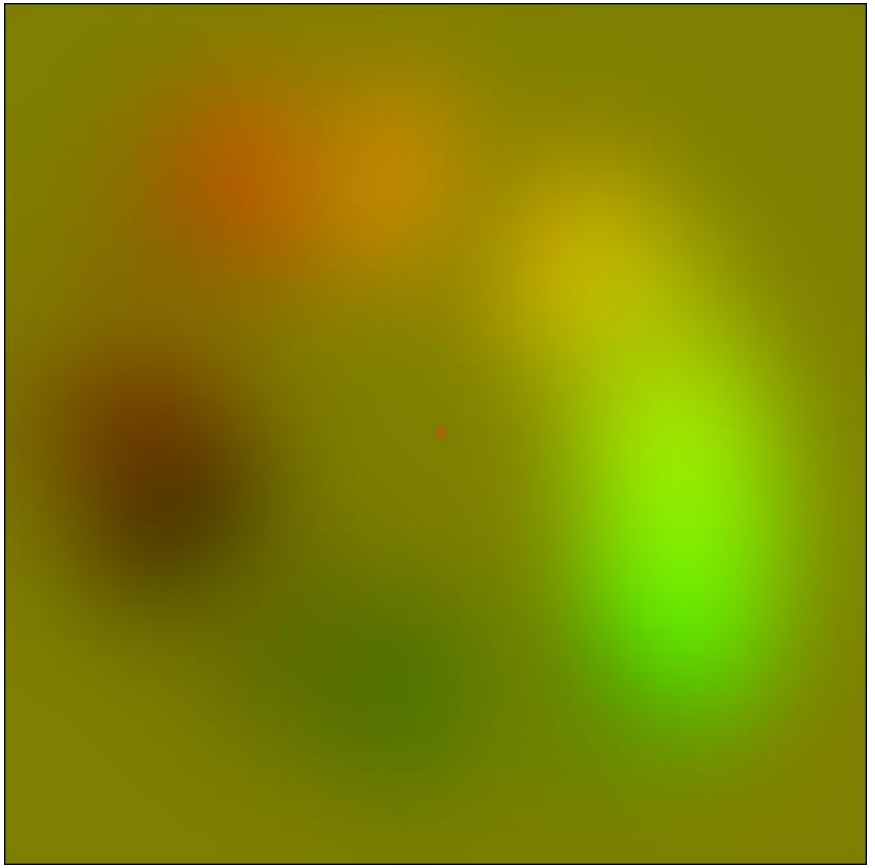
It have hard edges between points colors, so you need simple smoothing to fix that.



Smooth
Apply To = Other Attribute.
Attribute Name = N.
Cutoff Frequency = 10.
Smoothing Iterations = 100.

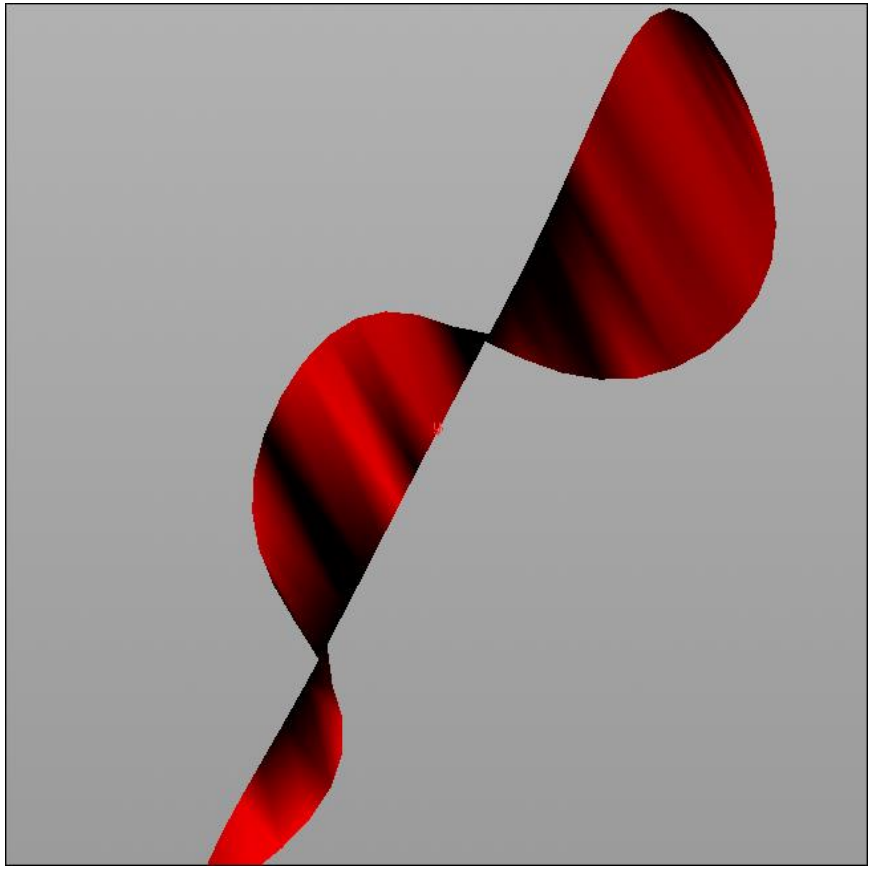
Optionally, using **Paint** tool, you can change normals length by point's colors, and get slower or faster flow in every point.



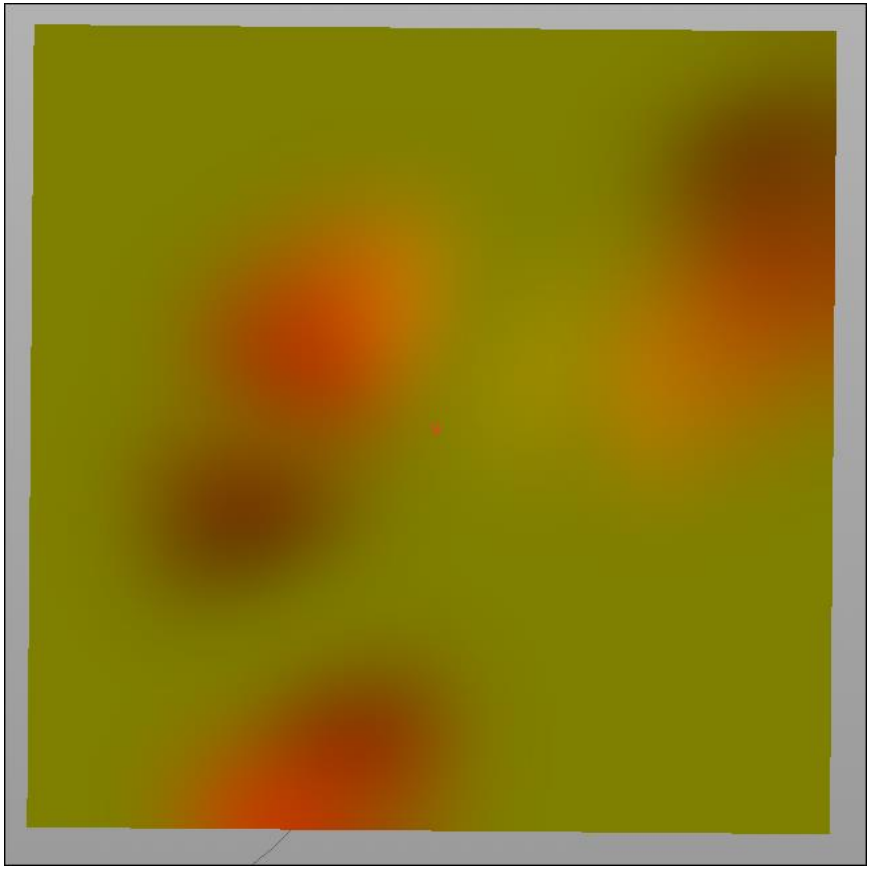


Working with opened curve, you need to close it before painting, because opened curve can't be painted.
So, return to curve object, check it's "Close" checkbox, go to grid network, paint curve color, and then open it again.

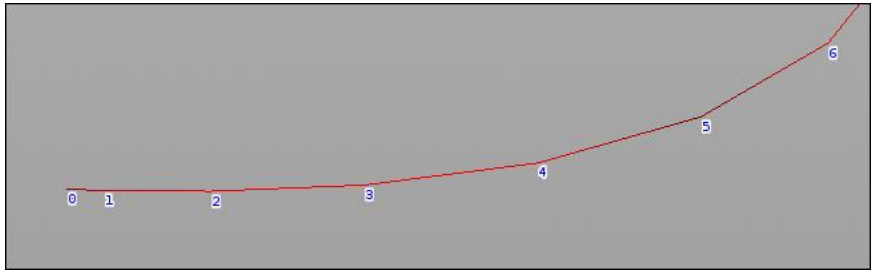
Closed curve with painted colors.

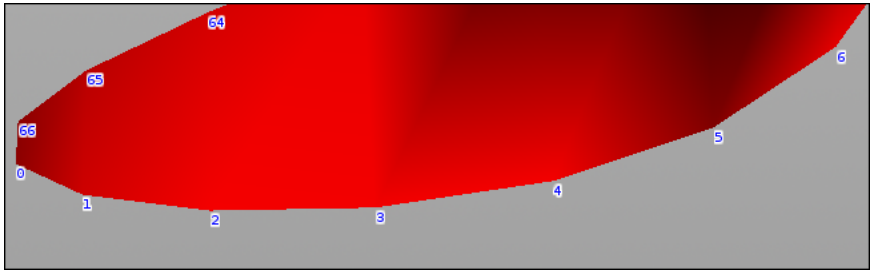


Final color with opened curve.

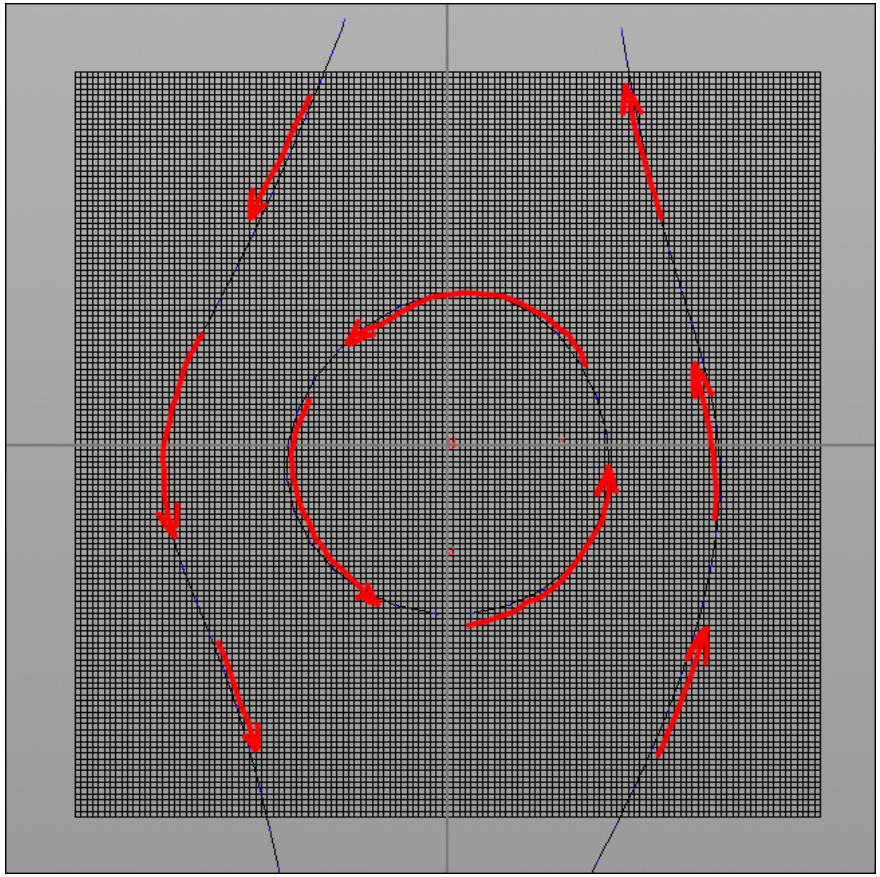


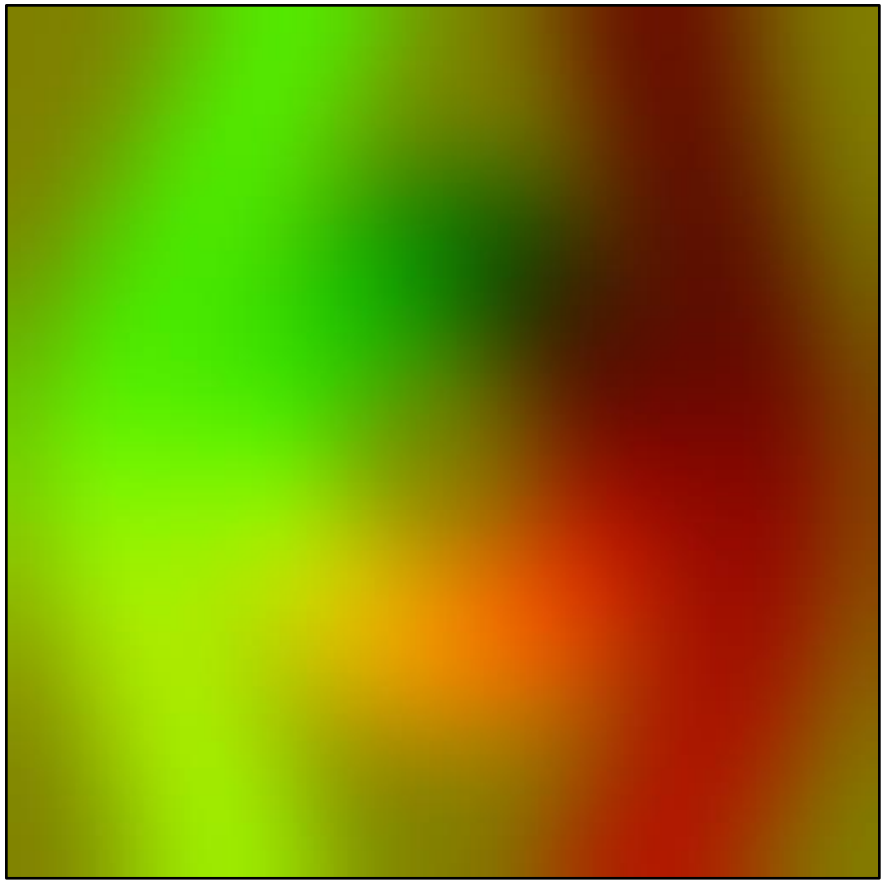
Don't worry about points order. Closed curve have additional points, which numbers not replace opened curve's points numbers.



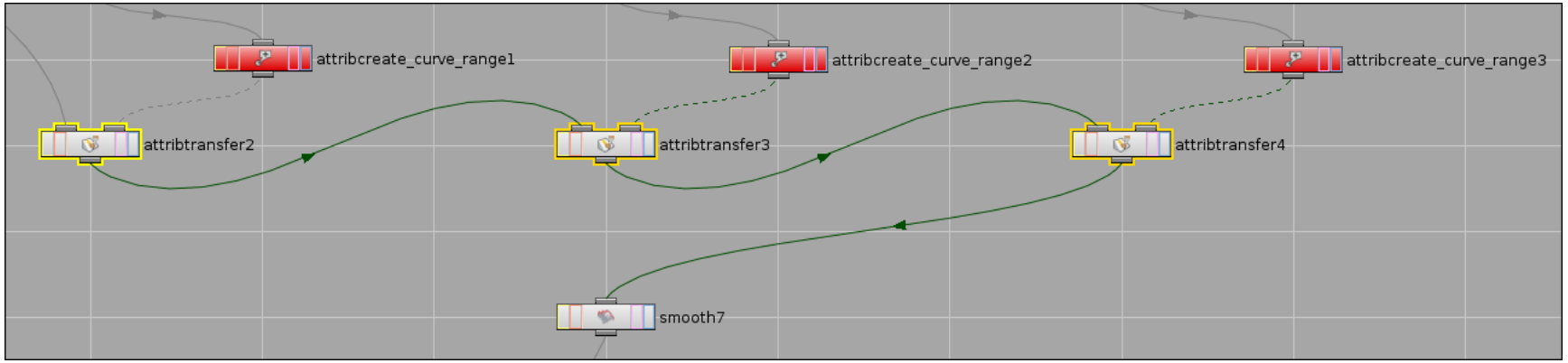


How this can be used on practice.
For example, I create 3 curves and setup it to get this kind of flow.





So, here I need to transfer normals from 1st curve, then from 2nd, and then from 3rd.



Final network.