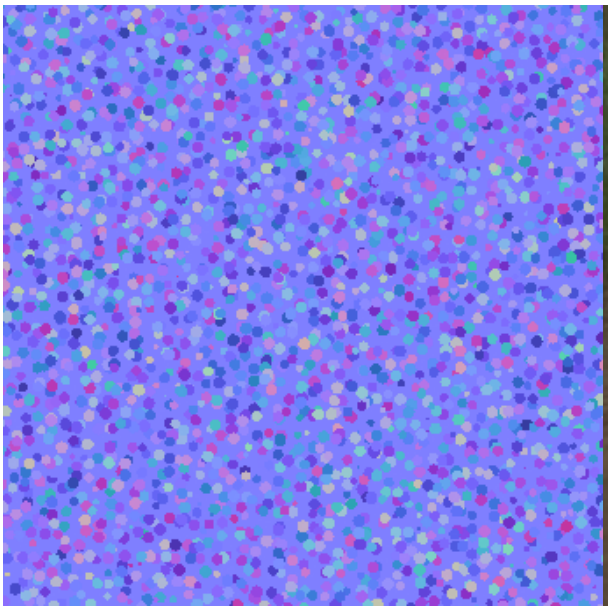
Flakes normal map



Overview

This texture creates a procedural flake normal map that can be used e.g. for car-paint-like materials. It is intended to be used as a normal map inside the bump slot of a material. Note that the map does not perform any filtering, so small flakes require a lot of AA samples to get clean.

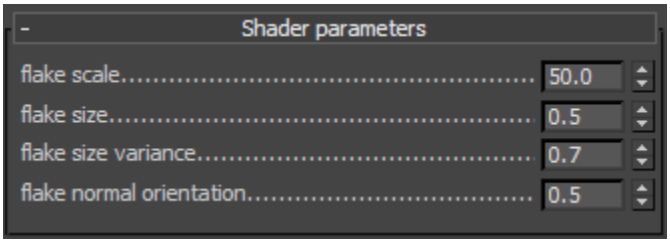
The shader code is here: flakes.osl; a sample scene for 3ds Max is here: flakes.zip

Parameters

Outputs

result - the flake normal map color

alpha - a black and white mask for the flakes



Inputs

flake scale - scales the entire flake structure up or down. Larger values zoom out of the map.

flake size - the relative size of the flakes.

flake size variance - determines the randomness of the flake size. 0.0 means all flakes have the specified size.

flake normal orientation - blends between the random flake normal and the smooth surface normal.

Shader code

Here is the shader code:

```
shader
flakes
(
        float flake_scale = 10.0 [[ string description = "Smaller values zoom into the flake map, larger values
zoom out" ]],
        float flake_size = 0.5 [[ string description = "Relative size of the flakes" ]],
        float flake_size_variance = 0.0 [[ string description = "0.0 makes all flakes the same size, 1.0
assigns random size between 0 and the given flake size" ]],
        float flake_normal_orientation = 0.0 [[ string description = "Blend between the flake normals (0.0) and
the surface normal (1.0)" ]],

        output color result = 1.0,
        output float alpha  = 1.0
)
{
        float safe_flake_size_variance = clamp(flake_size_variance, 0.1, 1.0);
        vector cellCenters[9] = {
                vector( 0.5,  0.5,  0.0),
                vector( 1.5,  0.5,  0.0),
                vector( 1.5,  1.5,  0.0),
                vector( 0.5,  1.5,  0.0),
                vector(-0.5,  1.5,  0.0),
                vector(-0.5,  0.5,  0.0),
                vector(-0.5, -0.5,  0.0),
                vector( 0.5, -0.5,  0.0),
                vector( 1.5, -0.5,  0.0)
        };

        point position = vector(u, v, 0.0);
        position = flake_scale * position;

        point base = floor(position);

        point nearestCell = point(0.0, 0.0, 1.0);
        int nearestCellIndex = -1;
        for(int cellIndex = 0; cellIndex < 9; ++cellIndex)        {
                point cellCenter = base + cellCenters[cellIndex];

                vector centerOffset = cellnoise(cellCenter) * 2.0 - 1.0;
                centerOffset[2] *= safe_flake_size_variance;
                centerOffset = normalize(centerOffset);

                cellCenter += 0.5 * centerOffset;
                float cellDistance = distance(position, cellCenter);
                if(cellDistance < flake_size && cellCenter[2] < nearestCell[2]) {
                        nearestCell = cellCenter;
                        nearestCellIndex = cellIndex;
                }
        }

        result = color(0.5, 0.5, 1.0);
        alpha = 0.0;

        if (nearestCellIndex != -1) {
                vector randomNormal = cellnoise(base + cellCenters[nearestCellIndex] + vector(0.0, 0.0, 1.5));
                randomNormal = 2.0 * randomNormal - 1.0;
                randomNormal = faceforward(randomNormal, I, randomNormal);
                randomNormal = normalize(mix(randomNormal, vector(0.0, 0.0, 1.0), flake_normal_orientation));

                result = color(0.5*randomNormal[0]+0.5, 0.5*randomNormal[1]+0.5, randomNormal[2]);
                alpha = 1.0;
        }
}
```