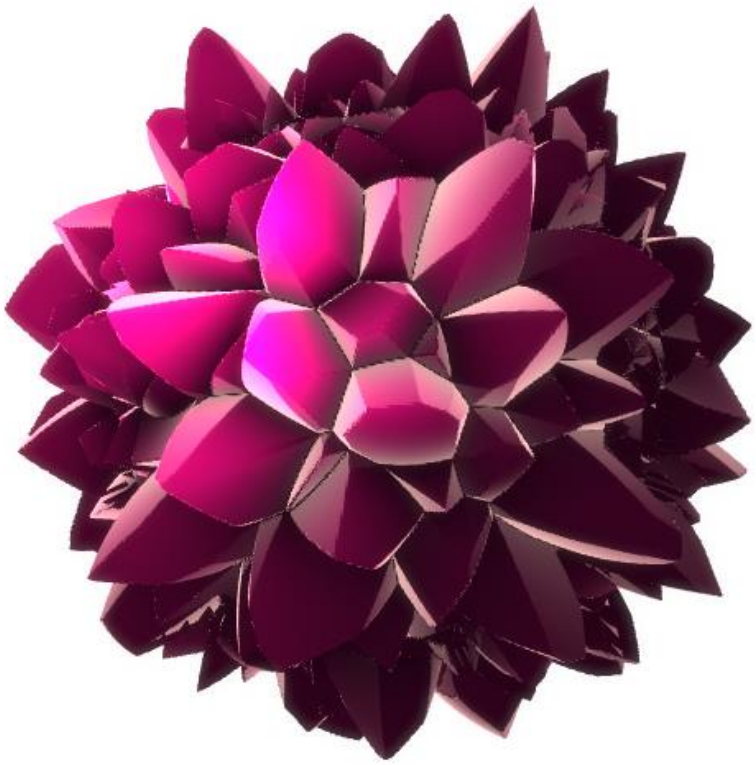
2020



# TNM084 PROCEDURAL METHODS FOR IMAGES

MARTIN MARKLUND

# Procedural Crystal Shader

The goal of this project is to create a shader program in Unity3D that deforms a tessellated sphere to give it a crystal-like appearance. The result is a shader program with several tweakable settings that allow the user to customise the characteristics of the crystals.

## Pre-study

Before the implementation the characteristics of crystals were studied. The goal was to define the most defining features that would help sell the visualisation. Early on, it was decided that the shader would aim to produce "uncut-gem" like crystals, rather than jewellery. During the pre-study, other attempts to create procedurally generated crystals were found. However, these implementations utilised node-based shader coding which differed from this project which would use HLSL.

The main defining features that were found during the pre-study could be narrowed down to:

- Sharp angles and surfaces
- Convex shapes
- Specular highlights
- Transparency (to a certain degree)
- Light scattering

These features could then be divided into two categories: shape alterations and lighting. This falls well in line with the structure of shader programs which utilises a vertex and a fragment shader.

## Implementation

The main implementation of the project was done in a Unity3D shader program. The shader has one vertex shader and one fragment shader. The shader can take user input via the Unity editor as shader properties. These properties are evaluated at run-time and can be altered at any given time to change the characteristics of the crystals. Apart from the shader program, a shader include file with utility functions was implemented. Finally, a small C#-script was implemented that fed the camera position to the shader. The following sections will describe which parts of the implementation that were done in the vertex shader, and which parts that were done in the fragment shader, respectively.

### Vertex Shader

The shape of the crystals was implemented in the vertex shader. Each vertex was displaced along its normal direction according to a noise function. The greater the value of the noise, the greater the displacement strength. In order to mimic the sharp and pointy crystals a suitable noise had to be used. From the pre-study, crystals were found to be convex. This greatly simplified the noise generation as well as vertex displacement, since if the crystals had to support concave displacements it would be impossible to do with a simple vertex shader, see Figure 1. You can think of it as "pulling" out the vertices from the surface; You cannot pull out a concave shape since there would not be any "empty" space below the concave parts. In the end, the noise function used was a three-dimensional Voronoi noise (Ronja Böhringer, 2018). (Initially the three-dimensional Cellular noise by Stefan Gustavson was used. However, there was some issues with "seams" in the noise which would make the vertex displacement overshoot.) The Voronoi noise (sometimes referred to as Cellular noise) is a quite simple noise that for each point describes the Euclidian distance to a site position, see Figure 2 a). Figure 2 b) shows the resulting displacement. The density of cells and the strength of the displacement can be fine-tuned with shader properties.

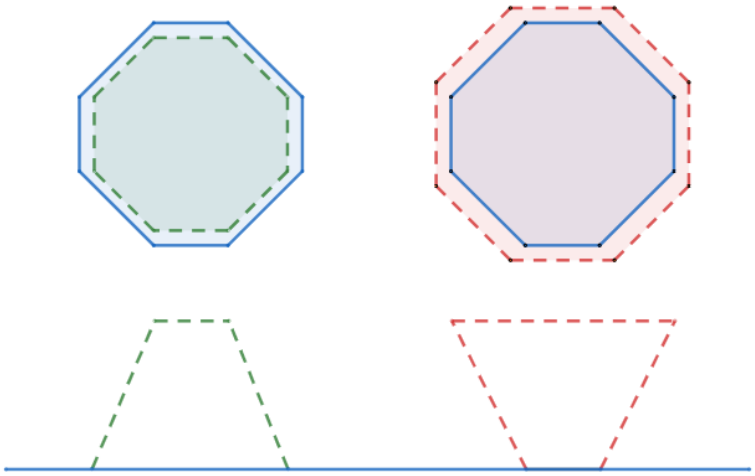


*Figure 1 Surface vertex displacement. The upper shapes show a top down view of the surfaces and the lower illustration depicts the surface from the side. The green dashed lines show a valid surface displacement whereas the red dashed lines show an invalid displacement, respectively. The invalid displacement would form a concave surface, which would obscure underlying vertices on the surface.*

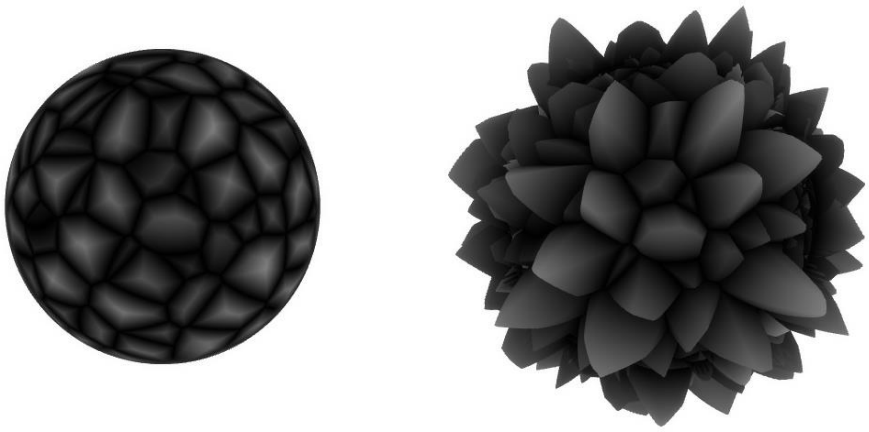


*Figure 2 a) Three-dimensional Voronoi noise, the brighter the pixel, the farther the distance to a site position. b) The vertices of the sphere shown in a) have been displaced along the normal direction according to the noise values.*

The noise implemented by Böhringer was written for a Unity3D surface shader program. When compiled, Unity's surface shaders generate a vertex and fragment shader. However, to fit into this project, the surface shader had to be converted into a vertex and fragment shader. To generate the noise, a random value was needed as well. Fortunately, Böhringer provided examples for generating seemingly random values within shader programs (Ronja Böhringer, 2018). Both the noise-function and the random number-generators were stored in the shader include-file. From the vertex shader, the local space position and world space position were passed to the fragment shader.

## Fragment Shader

In the fragment shader, the lighting characteristics as well as the colouring of the crystals were implemented. Arguably the most important line of code in the fragment shader recalculates the normal for each fragment. Since the vertices had been displaced in the vertex shader, any lighting calculations would be off since the normals would still describe the original vertex normals. A fast way of approximating the new normals is to use the cross product of the built-in shader functions *ddx* and *ddy* which calculate the partial derivatives in screen-space of a fragment. The drawback of this method, however, is that the new normals are in half resolution. Fortunately, for the purposes of this project, the visual impact of the lower resolution normals is acceptable. With the normals redefined, the rest of the lighting calculations could be implemented.

### *Specular Highlights*

As mentioned in the pre-study, one of the most defining characteristics of crystals is specular highlights. For this purpose, standard Phong shading with adjustable specular power was implemented. Lambertian diffuse shading and Unity's built in ambient sky lighting was also added as a base. Figure 3 shows the result of adding specular highlights to the crystals. To further improve the highlights, the Fresnel factor for specular highlights was added as well. The effect of the Fresnel factor increases the specular highlights when the light grazes the surfaces of crystals.

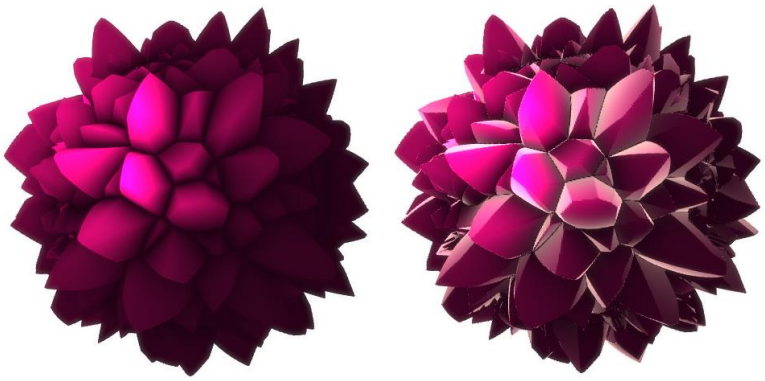


*Figure 3 To the left: no specular highlights. To the right: Specular highlights included.*

### *Transparency*

The second defining lighting characteristic of crystals is transparency. Some crystals are, to a certain degree, transparent. This is one of the features that give crystals their glass-like appearance. However, the transparency is very subtle and should not be as strong everywhere in the crystal; The thinner the crystal, the more transparent it should be. Since the size and shape of the crystal solely depends on the Voronoi noise, the same noise was generated in the fragment shader as well. (As it will turn out, the noise will be used for more purposes than just the transparency.) With the noise implemented, the transparency of a fragment could be described as $1 - (noise * T)$, where $T$ denotes the transparency factor which is an adjustable shader property. Figure 4 shows the crystal with- and without transparency. It should be noted that for most use cases, the transparency factor will be around 0.1. However, the transparency is in that case very subtle which is why, for demonstration purposes, the transparency has been increased.

### *Light scattering*

The final defining characteristic of the crystals is light scattering. When light enters material (in this case crystals) it is scattered before it exits the material. An easy way to visualise this effect is to place a finger on a flashlight. The finger will have a red glow and appear translucent. The same happens for crystals as well, but to another degree. Light scattering is not a simple effect, and usually takes a lot of computations to achieve. However, it can be approximated.

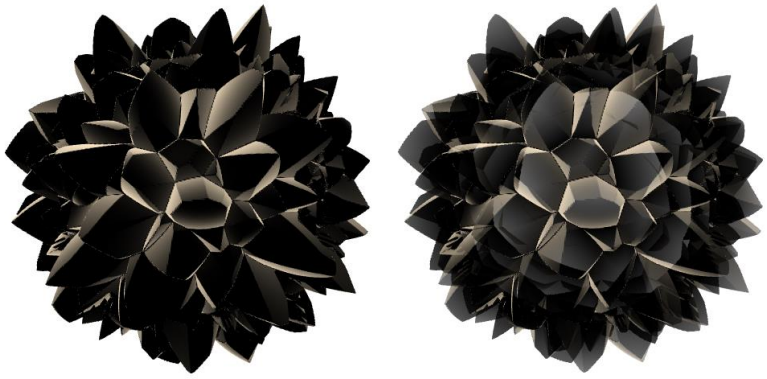


*Figure 4 The crystal with and without transparency. To the left: T = 0. To the right: T = 1.*

The approximation utilises the Henyey-Greenstein phase function, equation (1) (Henyey, 1941). The Henyey-Greenstein function is a phase function which describes how strong a signal is in a specific direction. In our case, we evaluate the strength of the lighting in its direction. However, the phase function in its original form uses exponentials, which are known to be slow shader functions. To tackle this problem, we can use the Schlick approximation of the function, equation (2), where $k = 1.5 \cdot g - 0.55 \cdot g \cdot g \cdot g$. The main advantage of the approximation is that it removes the exponent in the denominator of the original equation. The *g*-parameter defines how tight the scattering is, as shown in Figure 5. In the final implementation the g-parameter can be adjusted by the user.

$$p(\theta) = \frac{1}{4\pi} \frac{1-g^2}{[1+ g^2-2g\cos(\theta)]^{\frac{3}{2}}} \qquad (1)$$

$$p(\theta) = \frac{1}{4\pi} \frac{1-(k\cdot k)}{[1-(k\cdot\cos(\theta)\cdot(k\cdot\cos(\theta))]} \qquad (2)$$

*Figure 5 Different values of g and how it affects the light scattering. From left to right: g = 0, g =0.8, g = 0.9. The light is coming from behind slightly to the upper right corner.*

At *https://www.shadertoy.com/view/4ltGWl*, a comparison of the two equations can be seen. Note that the Schlick approximation still uses the *pow*-function, which somewhat defeats the purpose of the approximation. However, the result is still the same, and in this implementation the *pow*-function is not being used. The output of the phase function was then multiplied with the noise values to represent that the crystals are thinner at the top.

### *Colour*

The final part of the implementation regards the colour of the crystals. The colour is based off the noise, where it is brighter the greater the noise values are. The user can then define a main colour and a base colour, which will be blended through linear interpolation. It is also possible to use the noise to generate random colours for each cell.

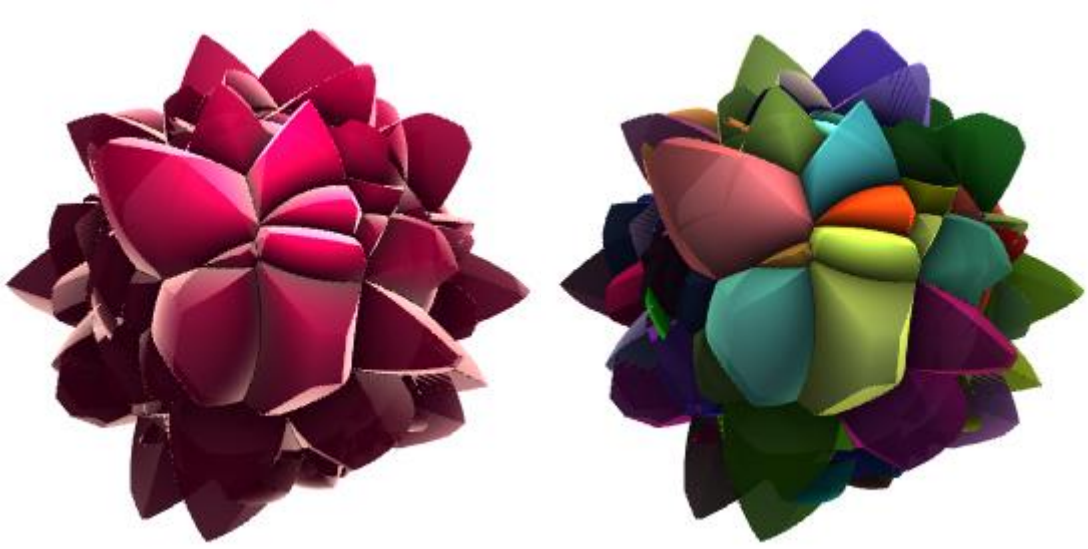
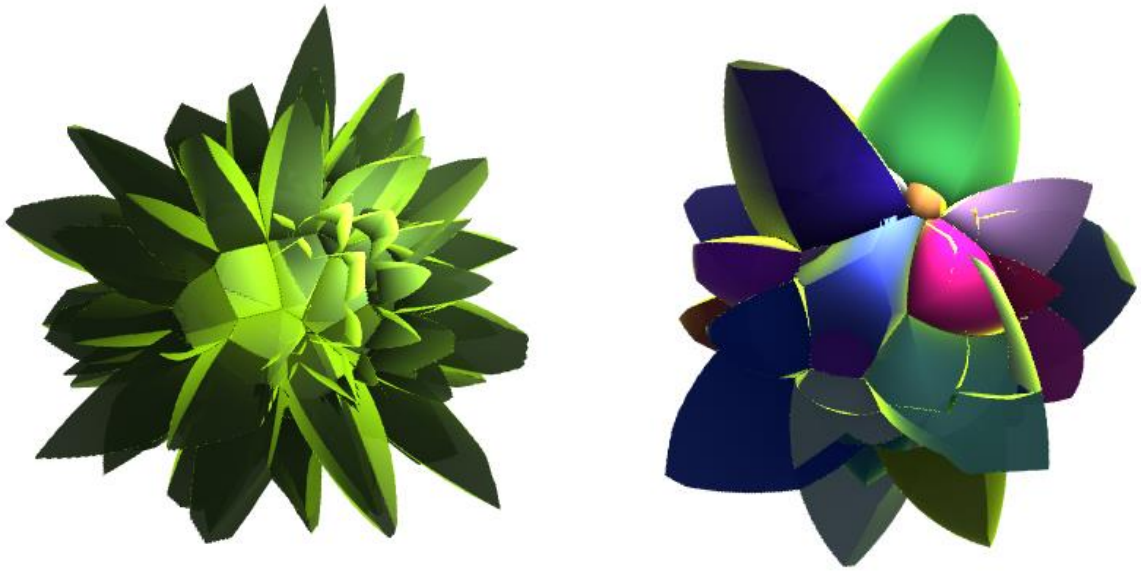
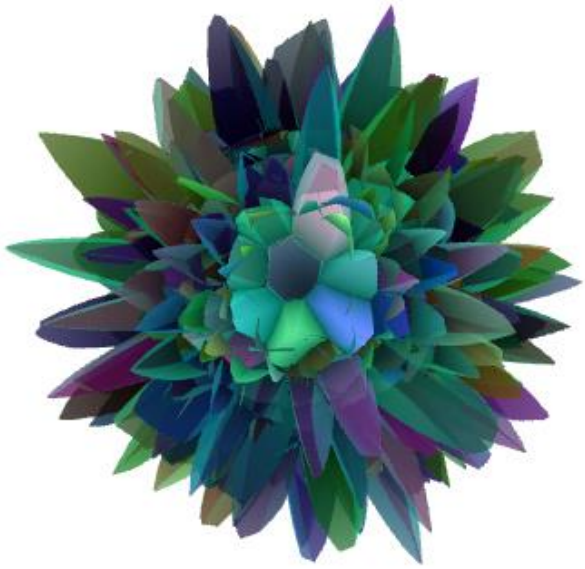


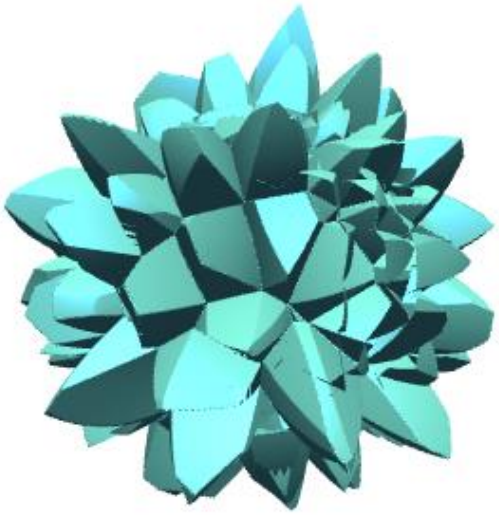
*Figure 6 To the left: User defined colours. To the right: Random colours based off the Voronoi cells. For the user defined colours, the specular highlights have been user defined as well. For the noise-based colouring, the specular highlights are based off the cell colours as well.*

## Results

Below some screenshots of different configurations from the final implementation are shown. The adjustable parameters are cell density, displacement strength, main colour, base colour, colour blend, specular colour, specular power, translucency and transparency.







## References


Ronja Böhringer. (2018, 09 28). *Voronoi Noise*. Retrieved from Ronja's Shader Tutorials: https://www.ronja-tutorials.com/2018/09/29/voronoi-noise.html

Ronja Böhringer. (2018, 09 2). *White Noise*. Retrieved from Ronja's Shader Tutorials: https://www.ronja-tutorials.com/2018/09/02/white-noise.html

Henyey, L. G. (1941). Diffuse radiation in the Galaxy. *Astrophysical Jounral, Vol. 93,* ,p 70-83.