Open World Procedural Generation

Dmitrii Sidorenko & Jerry Chang

Demo Video & Git Repo

We implemented procedural generation with Perlin noise in Cardinal3D, focusing on terrain/ landscape generation (LandscapeGen.cpp). 2D Perlin noise is implemented as a function *perlin(x, y, octave)* taking x, y, and octave parameters, where octave defines frequency of the noise. For each octave N, the noise is generated for the grid size of *land_size/pow(2, N)*. This defines the limitation for land_size to be a power of 2 and max octave parameter is limited to *log(land_size)*.

The heightmap generator is implemented in the function *LandscapeGen::generateTerrain(int maxOctaves)*. It generates maxOctaves grids with Perlin noise of the different octaves from 1 to maxOctaves and then outputs the terrain heightmap summing all octaves in each grid point x, y with exponentially decaying weights.

$$heightmap(x, y) = \sum_{oct=1}^{maxOct} perlin(x,\ y,\ octave)/2^{oct-1}$$

By looking at the generated map, we've found that max octave = 6 gives the look closely resemble to the natural terrain. Nonetheless, our implementation gives designers the ability to experiment with this parameter via a user interface control called "Number of Octaves." Designers can also set landscape size via "Land Size" in the control.

In addition, we calculated the grass density map based on the generated depth map to plant the grass accordingly. We wanted to model following natural facts:

1) Grass doesn't grow well on the top of the mountain where the temperature is too low
2) Grass doesn't grow well on vertical granite rocks

To model this two natural phenomena we introduced two parameters

1) Terrain sloppiness *Ts(x ,y)*
2) Absolute terrain level (grass threshold)

We evaluate terrain sloppiness *Ts(x, y)* in each point using standard deviation of points around points with coordinates (x, y). Grass density map can be calculated as below,

$$\sigma(x, y) = \sqrt{\sum(\mu - heightmap(x, y))^2}$$

$$Grass\ Density\ (x, y) = 1 - 1/\sigma(x, y)$$

We also built a simple visualization in Cardinal3D by constructing mesh from the generated depth map and grass density map (functions added in util.cpp). For grass visualization, uniform random sampling subjected to density value is used to place elongated green cubes to represent grass.

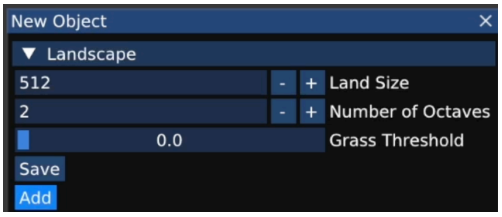

Fig. 1. Landscape and grass generation user control

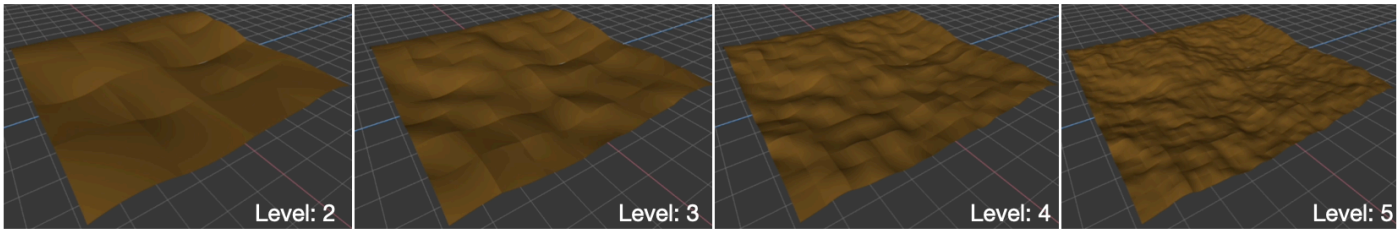

Fig. 2. Depth mesh visualization across different No. Octaves settings (land size = 512, no grass)

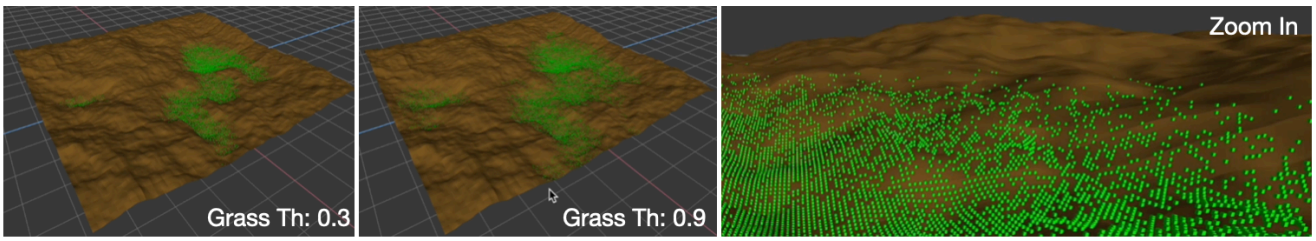

Fig. 3. Adjusting grass threshold and zoom in view (land size = 512, level = 6)

Then both depth map and grass density map are exported out as gray scale images that get imported to Unreal Engine to construct material procedural generation. With the aim to build fully procedural terrain, we implemented procedural material in UE5 using Blueprint visual programming language. We wanted to model following natural phenomena with our materials:

1) Color large slopes (almost vertical surfaces) in dark colors modeling rocks.
2) Color flat surfaces in yellow modeling sandstone.
3) Mix light green color between rock and sandstone modeling soil.
4) Plant grass on the flat surfaces sampling grass density from the density map exported from Cardinal3D.

Figure 4 and 5 show the Blueprint code of the implemented procedural material. There are 3 base colors: Dark rock color (layer 1), Yellow sandstone color (layer 2), Light green soil color (layer 3). The Lerp block interpolating between layers based on factors is supplied by the Blending block (Fig. 5). The idea is to blend layers (i.e colors) based on how close the normal to the surface to the Z axis in world coordinates. So this block takes normal in world space (VertexNormalWS parameter) and does dot product to estimate how close the current point normal in world space coordinates to the Z axis. We also introduced two additional parameters:

1) Offset (-1, 1) is an angle on a 2D sphere to start blending in layer color where -1 is 0 radians and 1 is pi/2.
2) Sharpness (1.0 to 10.0) is how sharp is the blending edge between two layers.

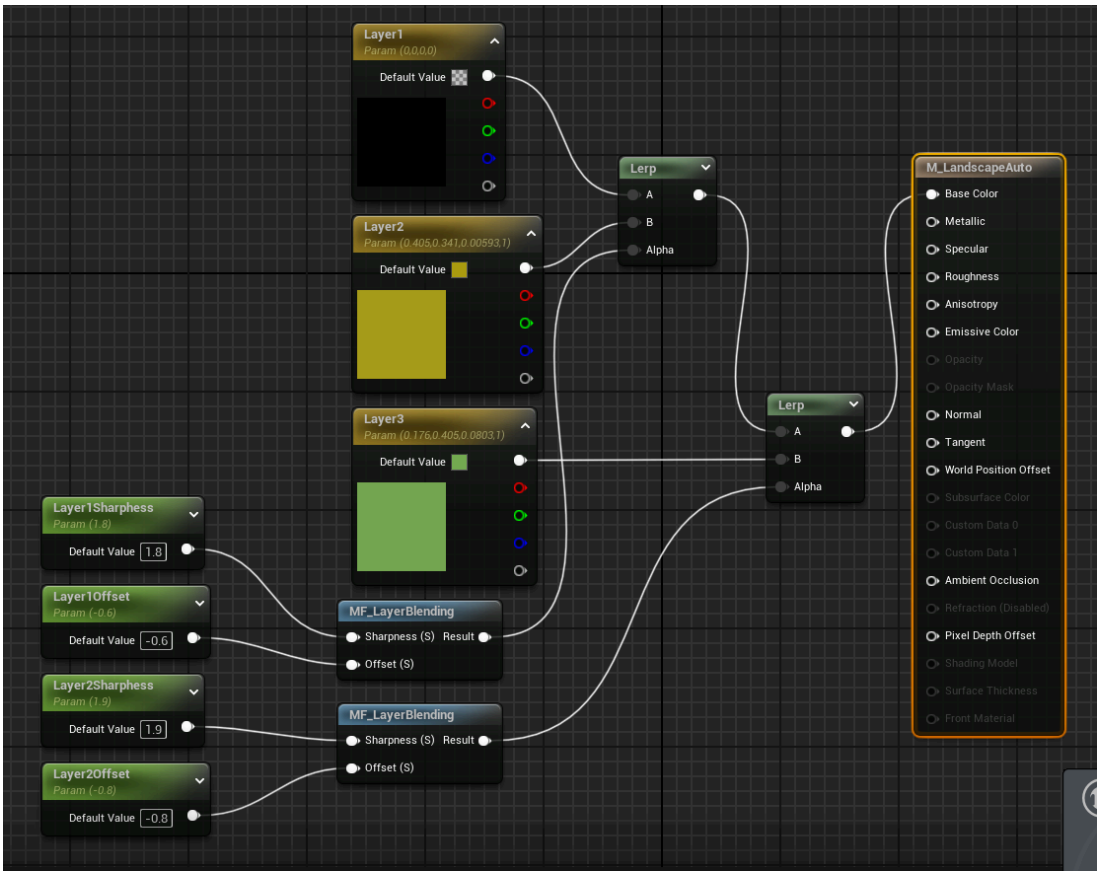

Fig. 4. Blueprint code of the implemented procedural material in Unreal Engine.

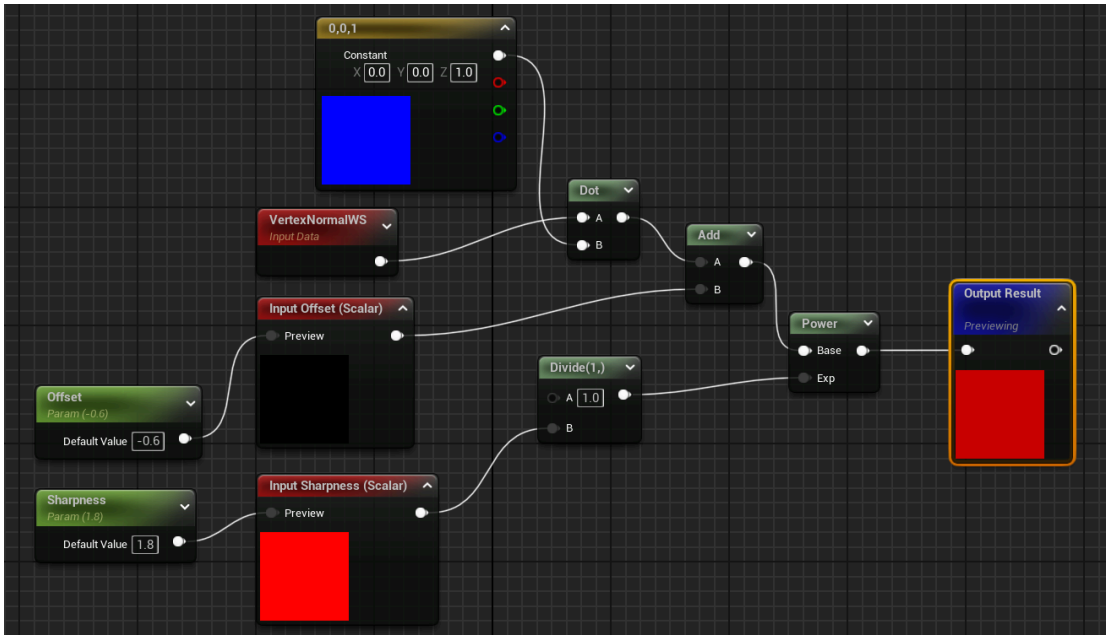

Fig. 5. Blending block implementation.

The second block of procedural material is planting grass (spawns 3D models) based on density sampled from images exported from Cardinal3D. This is shown in Figure 6.

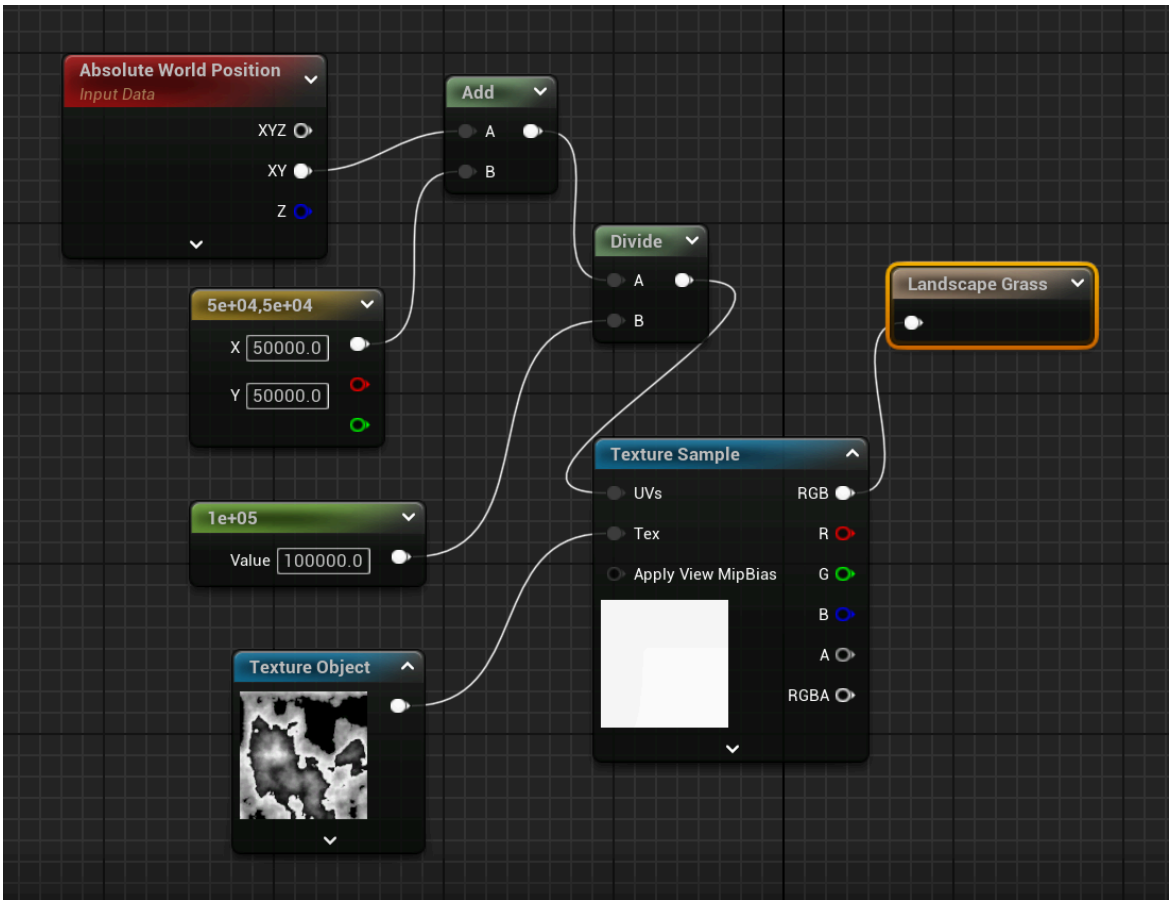

Fig 6. Grass planting sampler implementation.

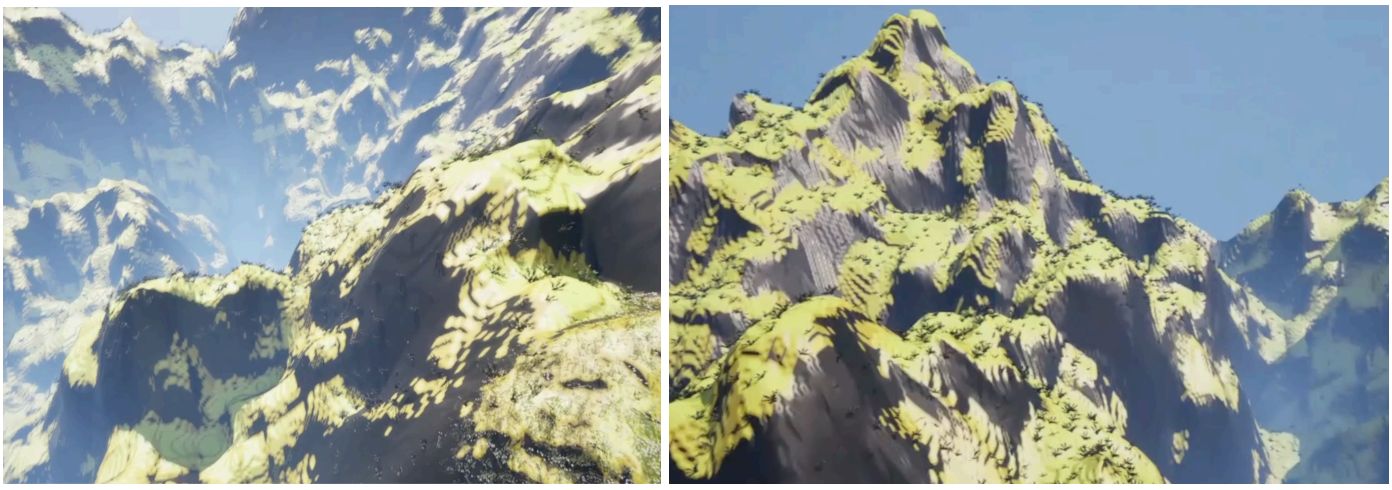

Fig. 7. Diving in the generated landscape with materials in Unreal Engine.