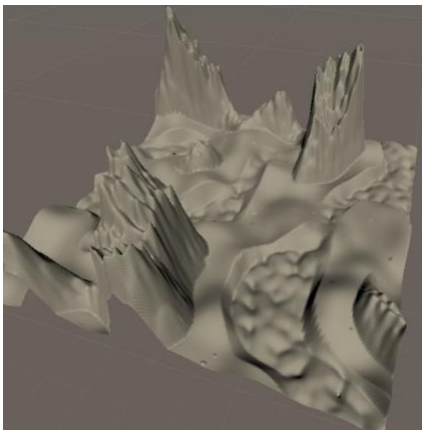
Procedural Content Generation with Perlin Noise



In this assignment you will be utilizing Perlin Noise generation and a hierarchical set of generation rules to create complex heightmap terrain.

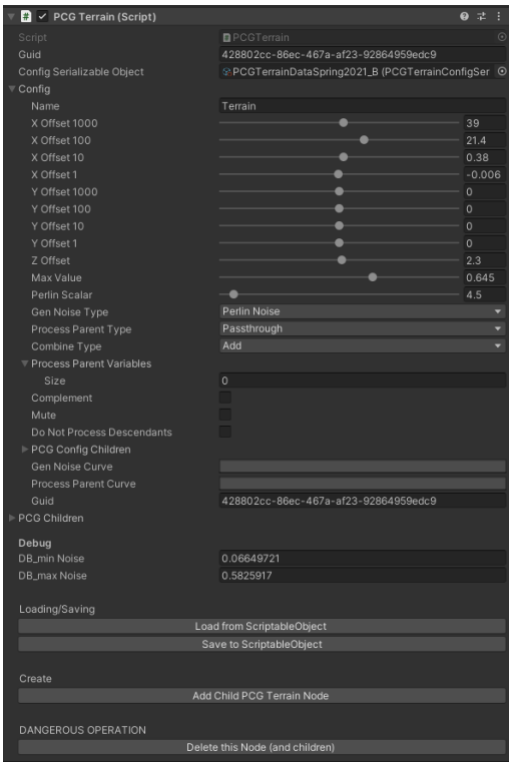
Perlin Noise is a type of gradient noise that is useful as a starting point for procedural content generation (PCG) of textures and terrain. Perlin Noise can be scaled to reflect random details at different frequencies. Noise can be processed and layered to create different effects.

For this assignment, you will use the PCG generator to design a terrain with three visually distinct areas (biomes or distinct geographic features). This will involve segmenting your terrain according to boundaries generated by Perlin Noise. For instance, you might make mountains in one, sand dunes in a second, and a canyon in a third. This configuration will be saved in a Unity ScriptableObject.

You will submit your designed terrain PCGTerrainData.asset (ScriptableObject), a readme.txt file describing it, and four (4) or more screenshots that best demonstrate what you developed. You can also share your screenshots on Piazza with your classmates. There are more details in the Submission Section.

Development Resources

A Unity Project (GameAI_PCG) is provided as a starting point. Open the project in Unity and then open scene, PCGTerrain. You will not need to “Play” the game at any point to create your terrain. All the work can be done in the Editor. But playing is good for taking a screenshot. You can also build and run the game to view the terrain (but not edit). Note that if you hit play (or run from build) you can exit by hitting Escape. You can move around with WASD and mouse-look, with mouse1 and mouse2 control elevation.



Building a PCG Graph

You will use the PCG generator (PCG Terrain component) to design a terrain with three visually distinct areas (or biomes). This will involve segmenting your terrain according to boundaries generated by Perlin Noise. For instance, you might make mountains in one biome, sand dunes in a second, and a canyon in a third. This configuration will be saved in a Unity ScriptableObject.

You will be declaratively defining rules/constraints on the noise generation to develop a compelling terrain scene. There are several workable strategies, but some methods are easier than others to organize things. Be sure to check out the related lecture video for tutorials.

The general steps involve configuring a node, then adding new nodes (via “Add Child PCG Terrain Node”) and then continuing the process of configuring and adding more nodes. Lastly, you need to save/serialize out to disk using the Save button. See below for more details.

PCG Configuration Details

The Name field is very helpful for keeping track of what layers do what. Be sure to change the name in the PCGTerrain Component and not at the top of the GameObject (PCGTerrain overrides the GameObject name).

The X/Y/Z Offsets (available in different levels of power for convenience) allow exploration of the noise space. This is useful for demonstrating how well your rules work if the terrain tiled out indefinitely. You can also use these sliders to find the best screenshots.

The MaxValue is used to limit the amplitude of generated noise.

The PerlinScalar allows “zooming” in/out on the noise (change the frequency of gradient changes).

The GenNoiseType allows for PerlinNoise, PerlinNoiseWithMappingCurve, and None. The mapping curve involves processing the generated noise with GenNoiseCurve (a manually configured function). If None is selected, then no noise is generated and a default value of 1.0 is used (scaled by MaxValue).

The ProcessParentType can be passed along unmodified (Passthrough), Inverted, processed/remapped with ProcessParentCurve, or a TrapezoidFunction (a bit like a bandpass and can help with crossfading terrain effects) The latter is especially useful for quickly segmenting noise for the purpose of creating regions/biomes. Note that the ProcessParentVariables is needed to define the bandpass region (4 values; the two outer values [index 0 and 3] define the start of the fade region and the two inner values [index 1 and 2] define the band). Anything outside the bandpass is mapped to 0.0. Anything inside is mapped to 1.0 (or faded from 0.0 to 1.0). You can also use the ProcessParentCurve to do the same thing, but it takes a bit more work to setup and is more expensive computationally to evaluate. It does have potential for smoother transitions though.

The CombineType selects how the current node's generated value is combined with the parent. Most commonly, you will want to either Add, Subtract, or Multiply. But there are a few other options such as NormalizeFromParentToTop that makes it easier to avoid clipping to the max ("top") elevation allowed by the heightmap (1.0).

Complement can be used to flip the final output value around (1.0 – outputVal).

Mute can be used to temporarily disable noise generation (value is 0.0).

DoNotProcessDescendant can be used to temporarily disable evaluation of children nodes.

The buttons at the bottom of the PCGTerrain component are also useful. You will need these for building your graph.

Load from ScriptableObject: Be cautious with this button! It will delete all children of the root node and reload/rebuild from the ScriptableObject. The load button is NOT smart. It is NOT aware of unsaved changes. The root and children will then match the ScriptableObject contents. Only available at the root of your graph.

Save to ScriptableObject: You must manually click this button to serialize your graph to disk (will be PCGTerrainData.asset with the default assignment). You should also save your scene from the Unity File Menu after saving the graph. Only available at the root of your graph. If you exit Unity without saving the scene (including crash or abrupt loss of power) then the ScriptableObject changes likely won't be preserved.

Add Child PCG Terrain Node: This adds a child node that has PCGTerrain configuration to the current node. This button is available at every node of your graph

Delete this Node (and children): Be cautious with this button! There is NO confirmation! This button deletes your selected node and children. It cannot be reversed and also affects your serializableObject too. This button is available at every node of your graph. You DO NOT need to delete nodes before Loading. Let the Load operation handle synchronizing things.

Validate: This operation will print “Valid!” to the console if everything is ok with the state of your ScriptableObject as compared to the data in the Hierarchy of your Scene. Otherwise, an error will be printed.

Other Considerations

Feel free to share techniques for building rules graphs on Piazza with classmates.

If you are having performance problems, you can reduce your terrain resolution. Select the Terrain component settings (gear) and scroll to “Heightmap Resolution”. Change to something smaller than currently. You might go higher for screenshots.

Grading

You will be assessed on whether you create three distinct areas that are isolated from one another within the rules you specify in PCGTerrainData.asset.

Key features we are looking for in your terrain graph file:

- Each of the three areas/biomes must be distinct, except for some crossfade to create smooth transitions.
- Each of the three areas/biomes must be generated from a root noise source by way of TrapezoidFunction or custom MappingCurve.
  You must either split the root noise source into three separate ranges or split hierarchically (e.g. split root noise source into two parts, and then split one of the parts in two).
- Each area/biome should have visually distinct heightmap patterns including multiple octaves of Perlin noise and intermittent features like rocks, stalactites, craters, etc. This will require multiple descendant PCG nodes in your graph under each area/biome.
- No area/biome should rely on clipping against minimum or maximum height values. These appear as perfectly flat areas with no details.
  (If you wish to create a water area such as a lake, you should either create the seabed or a turbulent water surface. If you make a seabed, it’s ok to add a translucent plane to show the waterline in your images if you like.)

Additionally, your submission will be checked to confirm that the required items are submitted and are formatted correctly (see below).

Submission

Submit your PCGTerrainData.asset (be sure you click “Save to ScriptableObject” in the PCGTerrain Component’s Inspector View and also save your scene from the Unity File menu), a readme.txt describing your design (include your name and a paragraph of text), and at least four (4) unique screenshots of your final terrain design and no more than ten (10). One or more images should be named overview_[n].[png/jpeg] where n is the index. The overview image(s) show all biomes in the same screenshot. The other three images are named [biome_name]_[n].[png/jpeg]. The biome_name is something like “mountains” or “swamp” as appropriate and n is the index. The image should prominently feature the described biome and may or may not show incidental parts of the other biomes. It’s ok to “scroll” around via the noise offsets to find good angles for each screen capture. For all

images, you can drop the index if you don't have multiple of the same type (e.g. "mountains.png" is fine if there is only one).

Submit all files individually (**NOT** in a ZIP file). It is a good idea to use a text editor to check your PCGTerrain.asset contents (YAML file format) and make sure that it matches what you see in Unity. You can also use the "Validate" option under the PCGTerrain Component in the Inspector Window of Unity.

Example submission files:

- PCGTerrainData.asset
- readme.txt
- overview_0.png
- overview_1.png
- mountains_0.png
- mountains_1.png
- swamp_0.png
- swamp_1.png
- hills_0.png
- hills_1.png