Planetary Terrain

Quadtree Planets for Unity

Getting Started

Your first Planet

1. **Create a Noise Module.** Noise Modules store a noise tree and can be created with the node editor. For now, we'll start simple. Create new a *Node Graph* (right click and create in project view), create a Generator and a Saving Node with RMB. Connect the Generator Node's output to the Saving Node. Choose a filename and press *Serialize*.
2. **Generate a heightmap.** You can generate a heightmap from a Noise Module with the built-in Heightmap Generator. Open *Planetary Terrain -> Heightmap Generator.* Press *Generate Preview* if you want, then press *Generate* (This will take a while, ~5min). Alternatively, you can assign the Noise Module directly to the planet later.
3. **Create a Material for the planet.** Create a new Material and select *Planetary Terrain -> Planet Surface Shader* as Shader. Assign your textures sorted by the height they should be used at, from lowest to highest. Use the *Texture Scale* field for tiling the texture.
4. **Setup the planet.** Create an empty GameObject and add the *Planet* Component. Assign your Material and Heightmap under the *Visual* and *Generation* Tabs respectively. If you didn't generate one previously, change *Generation Mode* to Noise and assign your Noise Module.
5. **Add Gravity and an Atmosphere (Optional).** If you want you can add the *Atmosphere* and/or *Spherical Gravity* Components. When using *Spherical Gravity*, make sure that your object has a Rigidbody, is tagged "Gravity" and to disable the standard gravity in *Edit -> Project Settings -> Physics -> Gravity*
6. **Play!**

API

The *Planet* and *Utils* classes provides some useful functions to convert from spherical to cartesian coordinates (or vice versa), to instantiate GameObjects on the planet and to calculate down vectors and rotations at a point on the planet.

Generation Mode

The *Planet* class can use different kinds of height data.

The simplest and fastest way is a heightmap. It yields the best performance, but the resolution can be too low when used with large planets.

Heightmaps are generated from noise, and it is also possible to do that at runtime. This has an essentially unlimited resolution, but calculating complicated noise at runtime takes a toll on performance and simple noise looks dull.

This asset also features a hybrid mode. A heightmap is used for the macro scale, and noise is used for smaller details. Performance is good because very simple noise can be used, but obviously using only a heightmap is faster. When using this mode make sure to turn the frequency up to 100-5000, depending on the size of the planet. You can also turn the octaves down for faster computation. The magnitude of the noise can be adjusted with 'Noise Divisor'.

Lastly, there is Compute Shader mode which is just like Noise mode, but the noise is computed on the GPU. This allows for many Quads to be split at once without mayor performance drops, if the target platform has a modern GPU and the CPU is limiting factor.

Heightmap Generator

The Heightmap Generator is a utility to create heightmaps from noise.

The Heightmap Generator generates three files – A TextAsset with the extension .bytes, and two normal .PNG textures. The textures are not needed for using the Planetary Terrain, but can be used for a copy of the planet in Scaled Space or be edited in an image editor and converted to a .bytes-heightmap with the Texture Heightmap to RAW utility.

The Generator features a 16-bit mode, 16-bit heightmaps have 65536 elevation levels as opposed to 256 of an 8-bit heightmap. They are recommended if size isn't an issue, 16-bit heightmaps are double the size of 8-bit heightmaps.

You can also use a RAW heightmap as input data for the generator. It will generate a color texture and a heightmap texture.

Node Editor

The node editor is a utility to create Noise Modules by combining different types of noise. The Noise Module can then be assigned to a planet directly or used to generate a heightmap.

To create a new Node Graph, right click in the Project tab and select *Create -> Node Graph.* Double click on the Graph to open the editor. Nodes can be created with RMB. The noise graph always starts with Generator Nodes, then Operator Nodes can be used to combine multiple Generators and to create more complex surfaces. The final node is always the Saving Node which can serialize the noise graph into a Noise Module or a Compute Shader. The graph used for the earth-like example planets is included as an example.

GPU terrain generation

As mentioned previously, Planetary Terrain supports generating terrain on the GPU instead of the CPU. Since GPUs are very optimized for parallel computing, you can easily split tens of Quads simultaneously on a modern GPU, compared to 2-8 on the CPU.

The workflow when using GPU terrain generation is very similar to using normal noise modules. The Saving Node can serialize your Noise Graph into a Compute Shader, which you can assign to your planet or the Heightmap Generator when selecting Compute Shader mode, just like a normal Noise Module.

One weakness of GPU terrain generation is the Select operator. In the standard CPU implementation, it uses conditionals (if, else) for which GPUs aren't optimized. The Saving node will not generate conditionals and both paths of the Select Node are always computed, even when one of them is not used. Because of this, it can still be beneficial for performance to use conditionals of the GPU. This will have to be done by hand though, for an example look at the Compute Shader of the GPU Terrain Generation example scene.

Unfortunately, the Remap operator is not supported on the GPU. For lower level access you can edit the GetNoise-function in the Compute Shader directly.

Noise Parameters

Seed: Changes the output of the noise function.
Octaves: Amount of detail, or how many times the noise is layered over itself with a higher frequency. This should be higher for larger planets but can be decreased when increasing frequency.
Frequency: Amount of change per unit distance. Increase for bumpier surface.
Lacunarity: Multiplier that determines how quickly the frequency changes for each successive octave.

Foliage

You can generate foliage in the form of meshes and grass with the included grass shader on planets.

Using Foliage:

1. **Enable *Generate Details*.** Then select the biomes/textures you want to generate foliage in, or alternatively enable *Generate Foliage in every biome.* If your planet will rotate in-game, enable *Planet is Rotating.* If enabled, all points will be generated in a single frame, so performance will be worse.
2. **Set points per Quad.** Points are used for grass and meshes, so you should use a lot more when generating grass compared to just meshes. It also depends on the size of quad levels you want to generate foliage in.
3. **Generate Grass?** If you want to generate grass, enable it. For now, you can use the included Grass Material under "Planetary Terrain/Materials".
4. **Set Detail Level and Detail Distance.** Detail Level sets the level at and after which foliage is generated if the distance is smaller than Detail Distance.
5. **Add Detail Meshes and Detail Prefabs.** Now you can add other objects that will be spawned. Detail Meshes have better performance, because they don't require a GameObject. That also means that you cannot generate colliders for the object, or spawn complex objects with multiple meshes. If you want to do that, use a Detail Prefab.
6. **Set Detail Objects up.** Select what fraction of the generated points is used for the object, and how far it is offset up. For meshes you also need to set the scale, material and if you want to use GPU instancing (recommended if Material supports it).

Scaled Space

Scaled Space is a layer in the scene where everything is scaled up 100,000x by default. It can be used for a copy of the planet that is shown when far away, where computing the actual quadtree is unnecessary, or for moons and far away planets.

Using Scaled Space:

1. **Create a second Camera.** Also create a new layer called 'Scaled Space' and set the culling mask so that it is the only layer visible to that Camera.
2. **Add the *Scaled Space* Component.** Assign it to the main planet.
3. **Change the Main Camera's Clear Flags to 'Depth only'.** Remove the Scaled Space layer from the Culling Mask. Also make sure that the Main Camera's Depth is higher than that of the Scaled Space Camera.
4. **Generate a Heightmap.** Use the Heightmap Generator to generate a heightmap. The alpha value of the colors will later set the smoothness on the copy in Scaled Space, so if you want a reflective ocean, turn it to zero on all colors except the ocean color.

5. **Create a new Material.** Add texture and normal map from the Heightmap Generator. Set the Smoothness Source to Albedo Alpha. A normal map can be generated from the heightmap in Unity's texture import settings. It is also recommended to increase the max size to 8192 for both textures.
6. **Enable 'Use Scaled Space' and 'Create Scaled Space Copy' in the *Generation* tab.** Assign your Material in the *Visual* tab.
7. **Adjust 'Scaled Space Distance' to set when the planet switches to Scaled Space.** You can also lower 'Visibility Sphere Radius Mod', then quads in the background will be replaced with your Scaled Space copy.

Creating planetary systems

You can create planetary systems by moving and rotating planets, their Scaled Space copies will be moved along with them in Scaled Space. Two simple scripts, *Rotate* and *RotateAroundPlanet,* are included, along with the *System* example scene.

Floating Origin

The asset also features a floating origin system, it allows for very large planets. The origin is always kept close to the player, where the most precision is needed. You can use it by adding the *Floating Origin* Component to your Planet and assigning your Main Camera and Transforms that should be shifted when the threshold is exceeded (e.g. buildings on the planet).

MSD/Terrain bumpiness threshold

In addition to the linear distance threshold that every planet uses you can also use a quad's mean standard deviation (MSD) (basically bumpiness) to define whether it should split or not in a given situation.

Simply enable *Calculate MSDs* in the *General* tab of your Planet. Then you can set an MSD threshold for each one of your detail levels. Usually everything but the last one to three levels are set to zero. For those levels there is no MSD threshold and even perfectly spherical (flat) quads would be split. The last three levels can for example be set to 0.2, 1 and 10, then only very bumpy quads with an MSD of over ten will reach the last detail level and a certain degree of bumpiness is required for the second to last and third to last levels.

Biome Mapping

Normally the calculated height values are not only used to displace the vertex but also to texturize it. This works fine but doesn't allow for different biomes at the same height. To still achieve this a biome map, or override heightmap, can be used.

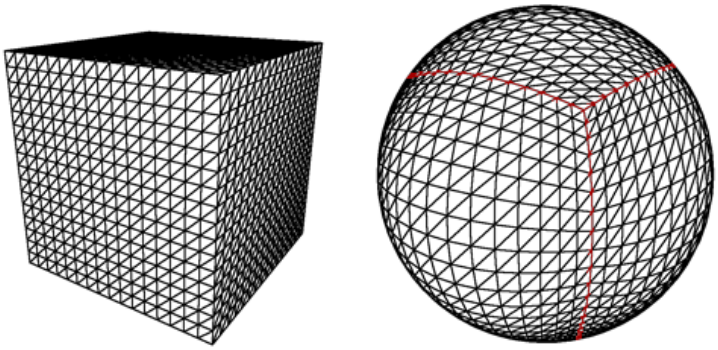
When using a biome map, the normal heightmap or noise is still used to displace vertices, but the height data used for texturing now comes from the override heightmap that can be edited without influencing the terrain's height at that position.

Explanation

This terrain system is based on the quadtree LOD system. It's quite simple – you start with a grid of 2x2 quads. When you need a higher level of detail in a certain area, you can subdivide a quad into four smaller ones, each of which can then be subdivided again.

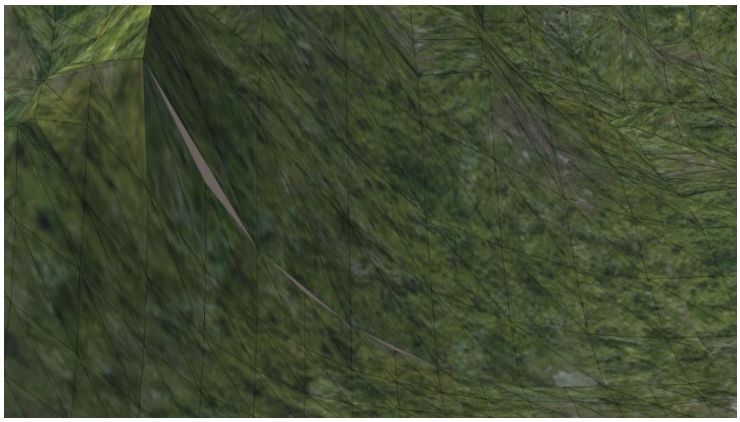
To use it for a planet, the planar quadtree has to be spherified. This is done in three steps:

1. A cube is formed from six quads. It always has the same size regardless of the planet radius, as it is scaled up later (this is handled by the *Planet* class).
2. Every vertex is normalized. This forms a unit sphere (handled by the *Quad* class).
3. Every vertex is displaced by a height value read from the heightmap or noise, then the vertices are scaled up to the radius (handled by the *Quad* and *Planet* classes).

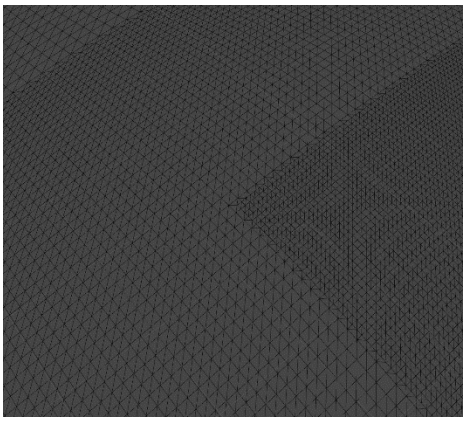


Cube before and after spherification.

Unfortunately, cracks appear at edges where quads of a higher level of detail border ones with lower a lower one. These cracks can be removed by deleting every other vertex on the quad with the higher LOD (quad edge fan).



Cracks



Quadtree Edge Fans

This however, requires knowing the neighbors of each quad to check if edge fans are needed. This process of finding neighbors is complicated by the spherification of the quadtree. In this asset it is handled by the *QuadNeighbor* class. An explanation of how it works can be found in the code. If a neighbor is of a lower LOD, the quad changes its own mesh to one with the right configuration of edge fans by changing the triangle array.