Etherea1 for Unity3D

Manual

INTRODUCTION

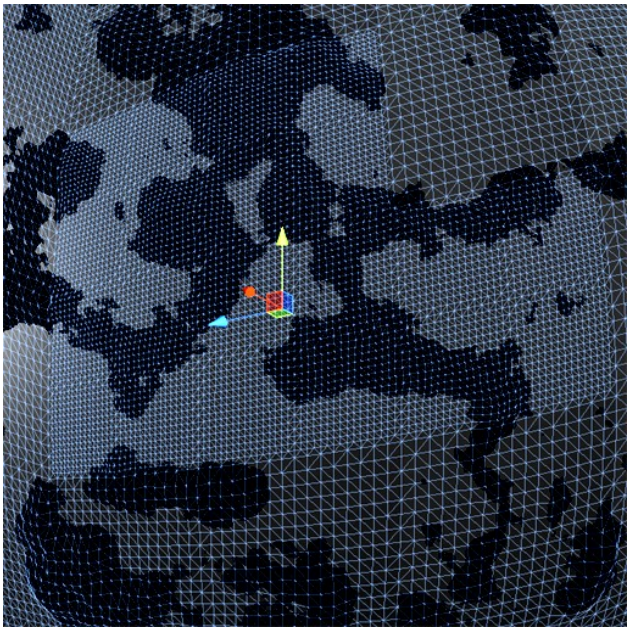
As you may already know, brute-force rendering of full scaled planetary bodies in realtime is still a challenge for current hardware, as the amount of triangles and memory are immense, easily surpassing the available videocard and even CPU memory and processing power. A single Planet in moderate detail will easily require many gigabytes of storage space, and even the most powerful GPU will have quite a hard time to render that amount of triangles in realtime.

Render tens of Planets using brute-force is just an impossible task for current mainstream hardware, and probably for some generations ahead. The amount of memory and processing power explodes too fast for a simple brute-force approach to work appropriately.

So, clever algorithms are needed for this specific task.

Etherea1 for Unity3D is a Procedural Planets generator which allows the creation of a virtually unlimited number of Fractal Planets, all with unique properties and characteristics, and their rendering in realtime. All Planets generated by Etherea1 are fully visitable and explorable. To accomplish this task, a number of techniques are applied in a way that processing power is spent only at places that are near the Camera, so places that are far from the Camera will have low detail and low use of memory/processing.

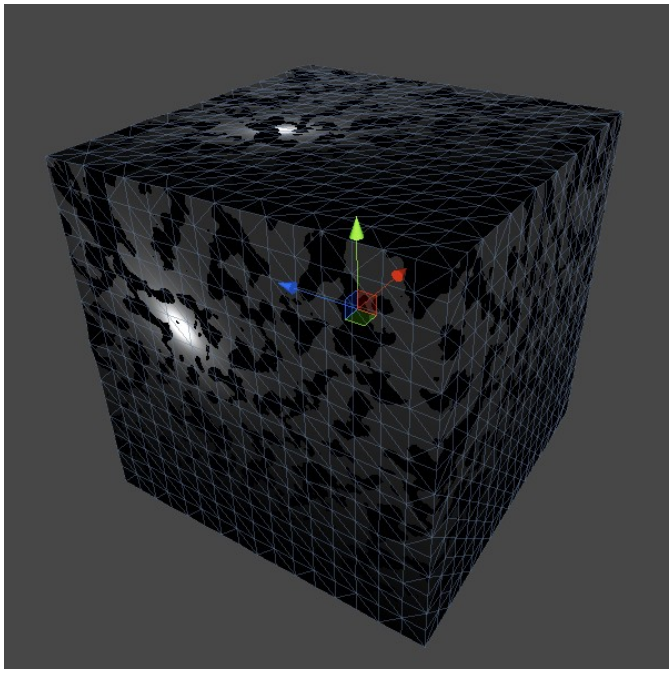
Simplifying, the main idea behind Etherea1 is a new variation of an old technique called Chunked Lod Quadtree, with a special property: Etherea1's Quadtree is split/rejoined on demand and in realtime.



If you take a look at the above figure, you will notice that, where our view point is closer to the Planet, the mesh is more detailed than at the other places that are a bit more distant. This is basically what the Quadtree does: it splits the mesh into more and more details as we approach the Planet, but only where our Camera is. That is, around the Camera the mesh is more detailed, and as distance increases, the mesh at that place gets less and less detail, until it is reduced to just a few triangles --- as we don't need to see them anyway, saving a lot of processing power and memory.

As our Camera moves through the Planet, the Quadtree gets updated, discarding the mesh that is not needed anymore, and generating other meshes that are now needed, given the new Camera location.

This is a simplified explanation, just to give you an idea. Quadtrees are not new, but Etherea1's Quadtrees are special because they work in realtime (not preprocessed) and they are deformed to fit a spherical terrain, differing from traditional Quadtrees which are flat.



Internally, Etherea1 creates a group of six Quadtrees, one for each side of a cube. The vertices of the Quadtrees are projected to form part of a sphere, and together they form a complete sphere.

Initially, each Quadtree is initialized to level of detail 0, where they are just a coarse patch of vertices with low detail. When our camera starts approaching one of the Quadtrees, that Quadtree splits to level 1, generating more detail at that location, then if we continue approaching it splits to level 2, generating even more detail at that location, and so on, until the maximum split level which is configured for that Planet. In the maximum split level, we will be probably at the ground level of that Planet. If we start leaving the Planet, the Quadtree will rejoin to the previous level, discarding some details, and will continue discarding data until we reach level 0 again and its coarse geometry.

But this is only part of the story, as this explains only how it tesselates the geometry. But the terrain topology itself, from where does it come?

THE FRACTAL GENERATORS

Each of our six Quadtrees is internally associated to one of a group of shaders – one of the GenNoise*.shader – which are variations of Perlin Noise. Perlin Noise is a widely known technique for generating pseudo-random numbers and vastly used to generate heights for terrain topology. Given a set of parameters, these shaders will generate heights for that part of the Planet, and working together with our Quadtrees, the shaders will generate from coarse terrain (when far from the Planet) to more and more detailed terrain (when close to the ground).

One interesting property of Perlin Noise is that, although apparently random, they will always generate the same set of random heights if we do not change the initial parameters. That means that, although apparently random, our Planets will always look the same, every time we return to them for a new visit. They will only change if we change one or more Noise parameters (detailed ahead).

So, lets simulate how it all works together: at start, the six quadtrees are set to level 0, and a Fractal terrain at coarse level is generated for each. As we start approaching one part of the Planet, the closer Quadtree splits and calls the Fractal generator to generate more detailed terrain for that place. As we approach more, it splits again and generates even more detailed terrain using the Fractal generator, and so on. When we start leaving the Planet, detail is discarded and more and more coarse terrain is generated.

As explained before, this is done to avoid memory/processing requirements explosion. As we generate detail only where it is really needed, we can keep things manageable and still realtime.

All these details are fortunately handled automatically for us by the Etherea1 sub-engine. You only need to understand the basic concept behind it all, so you know how to better use the available features.

A QUICK START

Before we start explaining the available parameters, lets create a simple Planet:

Within Unity3D Editor, press CTRL + N to start a new, empty scene.

(1) Go to the menu

GameObject -> Create Other -> Ethereal Planet

to create a new Ethereal Planet.
Your new Planet will have a default radius of 64 units,
and no textures at all.

Notice that a warning appears in the Console, saying
"Planet Ethereal Planet has no Sun. Please select a Light
as the Sun for this Planet."

We need a light to act as the Sun light for the Planet.

(2) Go to the menu

GameObject -> Create Other -> Point Light

to create a Sun light for your Planet.

(3) In the Hierarchy Panel, click the Ethereal Planet to select it. In the Inspector, find the "Sun Transform" field, click it and select the Light as the Sun for this Planet.

(4) Select the Point Light and move it above the Planet.

(5) Select the Main Camera and move it so that you can see the entire Planet in the Camera Preview.

(6) In the Project Panel, find the zSample folder, open it, open the Scripts folder, and find the CCameraOrbit script. Drag this script to the Main Camera in the Hierarchy Panel.

This script has some basic flying controls to move the Camera through the space, like:

[W] accelerate;
[SHIFT] + [W] accelerate faster;

[S] reverse accelerate;
[SHIFT] + [S] reverse accelerate faster;

[MOUSE MOVE] rotate camera;
[A] or [D] + [MOUSE MOVE] rotate differently (roll locked);

[SPACE] speed decrease until stop;

(7) Select the Ethereal Planet again. In the Inspector, find the "Water Tex" field, click on it, and select the "water 512" texture included in the sample package.

You should now see a water in the Planet.

(8) In the Inspector, find the "Water Normal" field, click on it, and select the "723-normal" texture included in the sample package.

The water is now normal-mapped.

(9) In the Inspector, find the "Water Tex Tile" field, click on it, and change the value to 32.

The water is tiling 32 times now, giving it a more realistic scale relative to the Planet size.

(10) In the Inspector, find the "Base Tex" field, click on it, and select the "forest 2" texture included in the sample package.

You should see the terrain with a grass texture.

(11) Please save and run the scene now, and look around a bit.

You should see a basic planet with land and water.

(12) Stop the scene.

(13) Select the Ethereal Planet again, go to the Inspector, find the "Has Atmosphere" field, and enable it.

(14) Save and run the scene again.

You should now see a bright horizon and a sky degradee around the Planet.

Lets add a sun Billboard to this scene.

(15) Stop the scene again.

(16) Go to the menu

GameObject -> Create Other -> Plane

(17) In the Hierarchy Panel, drag the Plane over the Point Light object, to make it a child of the Sun. Make sure that the Plane is at coordinates 0,0,0 after dragging it.

(18) Select the Plane and change its Scale in the Inspector to 5, 5, 5.

The Plane will look bigger in size.

(19) In the Project Panel, within Resources -> zSample -> Materials, find the "Sun" Material, and drag it over the Plane in the Hierarchy Panel.

The Plane should now have a Sun Texture
(the Sun Texture is included in the sample package).

(20) In the folder zSample -> Scripts, drag the Billboard script over the Plane in the Hiearchy Panel.

Now the Plane will always look directly at the Main Camera,
when the scene is running.

(21) Please save and run the scene again.

THE PLANET PARAMETERS

Sphere

This toggle is only for educational purposes, and may be removed in the future. Disabled Sphere will make the Planet cube-like, with six flat Quadtrees. In principle, this has no practical use at all.

Wysiwyg ("What you see is what you get")

By default, when in the Unity3D Editor, if you zoom in to a Planet, the Quadtrees won't split, so you always have only the coarse level 0 as a preview of your Planet. If you enable Wysiwyg, the Quadtrees will start splitting even in Editor mode, giving you a better approximation of how the Planet will look like when you are near it. Please be aware that this behavior is more appropriated for runtime, and the Editor may slow down to the point of freezing Unity3D.

Sun Transform

The light-source that illuminates the Planet. Any transform will work, and if it is a real light its properties will be ignored. In a future update, built-in lighting will be supported, so in the future this parameter won't be necessary anymore.

Camera

The Camera which will be used to fly through the Planet. This information is needed for the Planet to know which locations to generate more or less detailed terrain geometry. If there is a Main Camera in the scene, it'll be automatically selected by default.

Patch Quality

Vertex grid resolution for each terrain patch. There are 5 resolutions available: Minimum (7x7), Low (11x11), Standard (17x17), High (21x21) and Maximum (33x33 vertices). The higher Patch Quality, the more vertices will be generated for the terrain topology. This option greatly affects performance.

Normal Quality

Resolution of the heightmaps and normalmaps. Visually this affects the overall lighting of the normalmapped terrain. The higher Normal Quality, the more illusion of higher resolution terrain throug normalmaps you have. There are 5 resolutions available: Minimum (64x64), Low (128x128 to 64x64), Standard (256x256 to 64x64), High (256x256 to 128x128) and Maximum (512x512 to 128x128). This option also greatly affects performance.

Radius

The Planet radius, which is in other words the Planet size. Unity3D has a zoom-

out size in the Editor limited to 32768, but Etherea1 will support Planets much bigger than that. For really huge sized Planets (lets say, 1000000 radius), you'll need to create them in runtime, as the Editor won't support such a size, but that will be considerably more complex to handle. Most people will prefer to scale down their Planets to sizes within the Unity3D range for easier handling, though. You can easily fit a Solar System with 10 Planets within the Editor range if they have a maximum size of 10000 radius, for example.

Temperature

Poles or even the entire Planet may be gradually covered by a snow texture, if you desire. To simulate frozen poles, you provide an overall Planet Temperature in the range of -2 to +2 which means totally frozen (-2) to no frozen poles at all (+2). Generally speaking, Planets which are near the main Star (eg: the Sun) will have higher temperatures and less snow at the poles, but Planets too far from the Star will have lower temperatures and more snow at the poles. Also, it's recommended that your Planet is rotated in a way that the poles are always above and below the Planet in relation to the Star body, because in real life the facing parts (central parts) won't freeze too easily as they get a lot of energy.

Water Level

Internally, the range of the fractal heights will be in the -1 to +1 range for Turbulence fractals and in the 0 to +1 range for Ridged fractals. You will set the Water Level of your Planet accordingly to the desired incidence of water. Any height below the Water Level will be flatten and rendered as water. For example, Water Level = +1 will create an entirely underwater Planet, as fractal terrain will never reach that height. If you want a Planet with no water at all, you will set the Water Level to -1 or even -2. Intermediary values will give you more or less water occurrence.

Height Scale

The internal fractal heights will be multiplied by the Height Scale, so if you select Turbulence fractal terrain and Height Scale of lets say 100, you'll get terrain heights from -100 to 100. For Ridged fractals, this will be from 0 to 100. Other parameters like “Contribution” will also affect the final heights of the landscape.

Normal Multiplier

This parameter affects the overall influence of the normalmaps (lets say, the contrast) in the lighting. The value of 1 will get a very sharp and aggressive influence of the normalmaps, and a value of lets say 32 will get a very small and weak influence in the lighting. You need to find an intermediary value which better represent the overall influence in the visual of your Planet, as the exact preferred value is very subjective.

Max Split Level

As said before, the Planet will start at level 0 which uses very small memory and processing power. As the camera approaches the Planet, it starts subdividing and generating more and more detailed geometry at the camera's location, also increasing memory consumption and processing power. This parameter will limit the maximum subdivision level the Planet can reach. A low maximum split level (3, for example) will consume very little memory and processing, but the Planet won't be very detailed when you reach its ground level. For higher max splits (11, for example), you get more detailed geometry at ground level at the cost of more memory and processor usage. Etherea1 does not limit the maximum level you can reach at all, so you could experiment with absurd maximum levels like 20, but given floating-point precision limitations, you will notice a lot of problems like camera wobbling, ugly normalmaps, huge memory usage (probably surpassing the available videocard memory), to the point of a possible system freezing. With more experience and some careful tuning of all parameters, you could start reaching more and more higher split levels with good visual quality, so this parameter is left uncapped for you to have full liberty for experimentation.

Size Split

This parameter determines how much close the camera needs to be to a given Quadtree patch to split into more detailed ones. Mathematically speaking this is the size of the patch multiplied by this parameter. A low value like 2 will only split to a higher level of detail when the camera is very close to the node, but then you'll get very noticeable pops when the split occurs, because you are very near to the patch when it splits. Giving a high value like 10 for example will give you very smooth splits with almost no noticeable transitions, but it'll split very early, certainly affecting overall system performance. Generally speaking, faster machines will handle higher values, slower machines will bettetr handle smaller values. A value like 4 is a good start for the average, but you need to experiment a bit to get the best for your target system and your personal taste. This parameter is directly related to the next parameter, and you need to adjust both to get proper behavior. If you fail to set both parameters correctly, the system will behave incorrectly.

Size Rejoin

This parameter is the inverse of Size Split. It tells how much far the camera needs to be from a Quadtree patch for the patch to rejoin and discard details. Rejoins tipically happen when we are leaving the Planet, returning to open space. This parameter must be always higher than Size Split, two more than that at minimum. For example if you have a Size Split of 4, then Size Rejoin must be at least 6 for the system to still work correctly. Personal taste counts here again, but here is a proportion suggestion:

Size Split = 4, Size Rejoin = 6
Size Split = 7, Size Rejoin = 10

Size Split = 10, Size Rejoin = 15

Enable Physics

If you enable Physics for a Planet, the terrain geometry will include a Mesh Collider which can be used to automatically handle Collisions against the terrain. This will greatly affect the overall performance, and you need to balance basically every other parameter to still get good performance from the system. Simply setting all other parameters to highest and enabling Physics will require a highend system, as this is very heavy on processing power. Please read Unity3D documentation to understand how to do proper use of Mesh Colliders.

Gravity Power

This is a experimental parameter which affects the gravity of the Planet. Unity3D built-in Physics does not handle spherical gravity at all. In the source package you received there is a preliminar support for this, inside DropObjAI.cs. This preliminar spherical gravity system will be revisited in the near future for a more solid example of how to implement this feature.

Min Physics Split Level

Enablind Physics for the entire Planet can get very processing costly. If you do enable Physics, you probably will want to limit it to happen only when the camera is very near to the ground, so you can set it here. For example, lets suppose you've set Max Split Level of the Planet to 10. If you set Min Physics Split Level to 0, your Planet will have Physics support from level 0 and above, so even when you're completely in open space (very from the Planet), it'll be processing Physics already, probably wasting processing power. So, you could for example set the Min Physics Split Level to 9, for it to start processing Physics only when your camera is close to the ground, which is, only those nodes near the camera will be spending processing power for Physics handling.

Min Shadows Split Level

This parameter is actually unused, but in the future it'll tell the system at which split level it'll start to generate shadow maps for the patches.

Filtered Heights

When enabled, the normalmaps will be bilinearly filtered so that they will look more smooth. Some older videocards may not include support for filtered normalmaps, and some people may even prefer non-filtered normalmaps, like oldschool texturemappers.

*** TO BE CONTINUED VERY SOON ***

Layers

- Size
- Element n
  - Layer Type
  - Noise Offset
  - Octaves
  - Persistence
  - Frequency
  - Offset
  - Lacunarity
  - Hpower
  - Contribution

Water Tex
Water Normal
Water Tex Tile
Water Speed

Snow Tex
Snow Tex Tile

Base Tex
Base Tex Tile

Diffuse 1 Tex
Diffuse 1 Tex Tile
Diffuse 1 Lut Layer
- Minh
- Maxh
- Slope
- Aperture

Diffuse 2 Tex
Diffuse 2 Tex Tile
Diffuse 2 Lut Layer
- Minh
- Maxh
- Slope
- Aperture

Diffuse 3 Tex
Diffuse 3 Tex Tile
Diffuse 3 Lut Layer
- Minh
- Maxh
- Slope
- Aperture

Diffuse 4 Tex
Diffuse 4 Tex Tile

Diffuse 4 Lut Layer
- Minh
- Maxh
- Slope
- Aperture

Shape Top
Shape Bottom
Shape Left
Shape Right
Shape Front
Shape Back

Has Atmosphere
Atmos Scale
Atmos Color 1
Atmos Color 2
Sun Intensity
Min Atmos Alpha
Horizon Height
Horizon Intensity
Horizon Power
Fog Intensity
Fog Max Alpha
Fog Height
Fog Near
Fog Far