MMTerrain Engine – Quick Start-

"Google-Earth" style terrain engine for Unity.

(Questions Not Covered? Email support@freeterrainviewer.com)

- **Unity Terrain Rendering For Large Tiled Sets:**

The MMTerrain Engine is designed to run virtually unlimited tiled terrain sets with a minimum of setup in Unity. This is mostly applicable for simulation and GIS applications that need to quickly render large terrains with an immediate level of detail as needed.

Provided that resource management and load-balancing are implemented on the client side (factoring in Unity pipeline limitations), it is also possible to use tiled data for flight simulation or constant-FPS applications that require smooth performance. A procedural game-style mode is also available as an optional bonus for artificial environments, though the intended purpose is for large geographic data that cannot be otherwise rendered in Unity at all.

Target Audience:

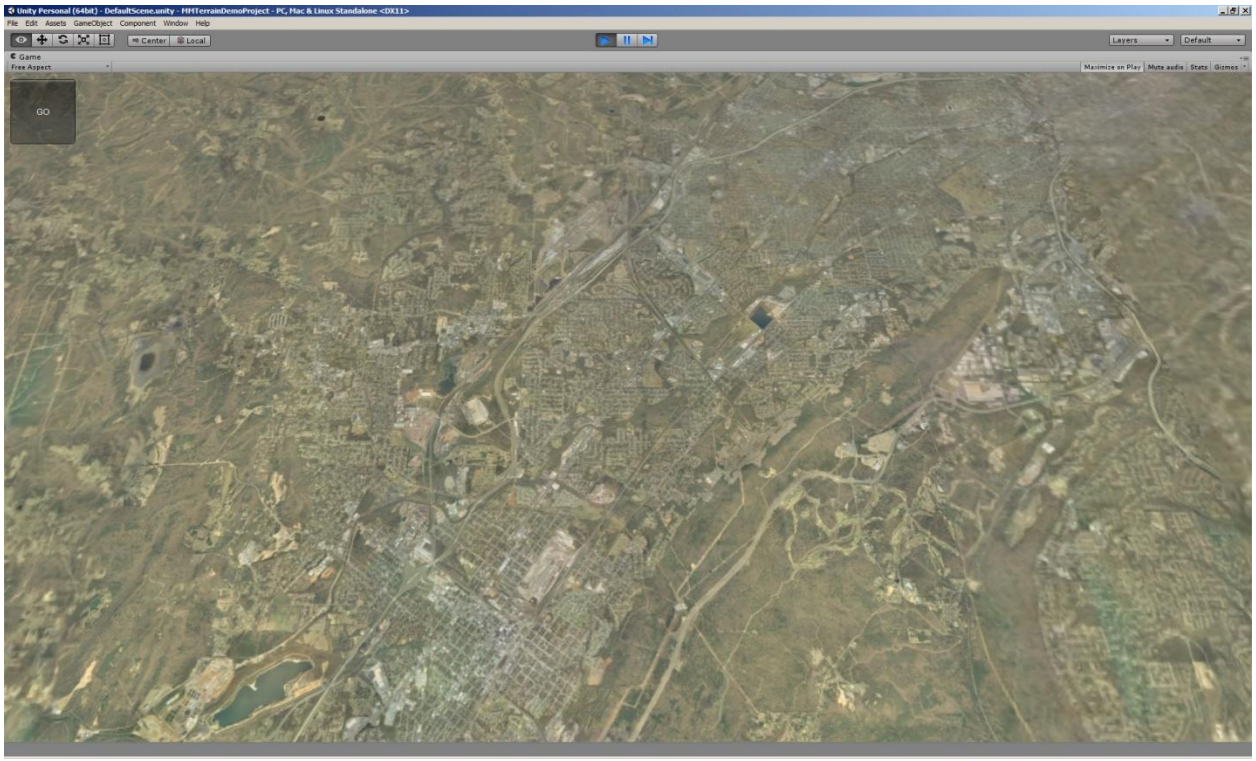
This package is geared towards programmers and engineers working with large terrains in Unity. Concept such as tiling, levels of detail, mip-mapping, heightfields, etc. should be understood. The default package comes with a basic runtime implementation (and optional tiling tool, for convenience) which provides a starting point for custom integration into the client application.

Custom terrain data can be used provided that it follows ideal settings, and metadata is correct. This is documented in the file “MMTerrainTechInfo.pdf”. Example databases can be downloaded and tested to verify setup.

How It Works:

Raw imagery and elevations are combined at runtime to provide a view of the terrain to the horizon in all directions, with immediate access to any local level of detail within range of the camera. View-dependent mesh and texture detail are handled automatically, based on settings.

Input can be procedurally generated, artificial, or based on real world data. The following image shows the entire city of Birmingham (from USGS). All areas are potentially visible from any vantage point, and any point of interest can be quickly examined in detail.



A live video of a fly-through (showing the monitor) can be seen at the following video link, this shows the scope of detail and transitions (within this particular dataset).

https://www.youtube.com/watch?v=63DBGjG2uwM

A video capture of the geometry culling and texture resolution can be seen here:

https://www.youtube.com/watch?v=ikkDk0kTOJg

Feeding Data Into The Engine:

The images tiles are never brought into Unity, they are stored externally and fed into the engine at runtime. In A Nutshell, once the initial parameters are set (such as the number of tiles, tile size, etc.), the core engine repeatedly requests Texture2D images from the client code, and the rest happens automatically. The example implementation includes a basic on-demand texture loader for disk or www.

The data is combined in realtime to produce a view-dependent mesh with dynamic level of detail (using a smooth-morphing approach for geometry, and a custom shader for combining textures). The output is a visible terrain that extends to a reasonable horizon in all directions, within the bounds of the terrain dataset.

Virtually No Pre-Processing:

The engine is designed to load arbitrary image sets and produce the result in realtime with virtually no pre-processing. However, in our testing, we have determined ideal settings for our GIS work (using data from USGS), and have developed a largely automated process to prepping data for the engine.

A scaled down version of this has been included with the package, as a matter of clarity and convenience, this makes it easier for others to ramp up quickly with the engine (and integrate data for testing) rather than get caught up in the tech details all at once.

Essential Extras:

The core setup and control is neatly wrapped into a single "BasicTiledTerrain.cs" script that can be attached to an empty game object. A text asset is used to define all parameters, though all settings could also be modified directly in the script itself.

A simple motion control script ("BasicMotionControl.cs") is also supplied that can be attached to an empty gameobject. This script intercepts KB/mouse input and updates the transform for the main camera, which is used to generate the terrain view. NOTE: It is assumed that virtually all client applications will have custom camera control, the motion control script is provided as a courtesy.

An in-house tiler tool is also supplied that (a) generates the text file with all settings, and (b) neatly organizes the images tiles into levels of detail. NOTE: The tiler tool is not part of the engine, it's a proprietary tool bundled with the engine for convenience. It works with geotiffs (using lat/lon and utm coordinates), or generic indexed image sets (World Machine tiled output, etc.).

Although these aren't required, they do provide a basic platform to get data integrated and running in the engine as quickly as possible. Further custom integration (such as alternate mesh rendering or source texture management) is possible depending on the client application, the basic implementation contains all the core ingredients needed to get started.

BASIC SETUP – QUICK START GUIDE:

The setup process is shown below, using a relatively small dataset from USGS.

<video link>

NOTE: The original example dataset can be downloaded here:

www.freeterrainviewer.com/CAHills.rar

The finished converted version of that terrain (ready to run) can be downloaded here.

www.freeterrainviewer.com/CAHillsConverted.rar

NOTE: The default setup also comes with a text asset for running this completed terrain directly from the server. This can be used immediately, out of the box.

Converting and/or running this terrain on your local setup will verify that everything is working correctly, and help to clarify the process.

Tech Info:

The default setup provides a basic platform to get up and running with terrain data. It is expected that the script and shader may be modified to accommodate the client application.

A look through the *.cs script will clarify how the textures are being requested/delivered, and the steps required to get the visible mesh. Integration with client code will be dependent on the application.

The core engine and critical metadata are described in the document “MMTerrainTechInfo.pdf”.