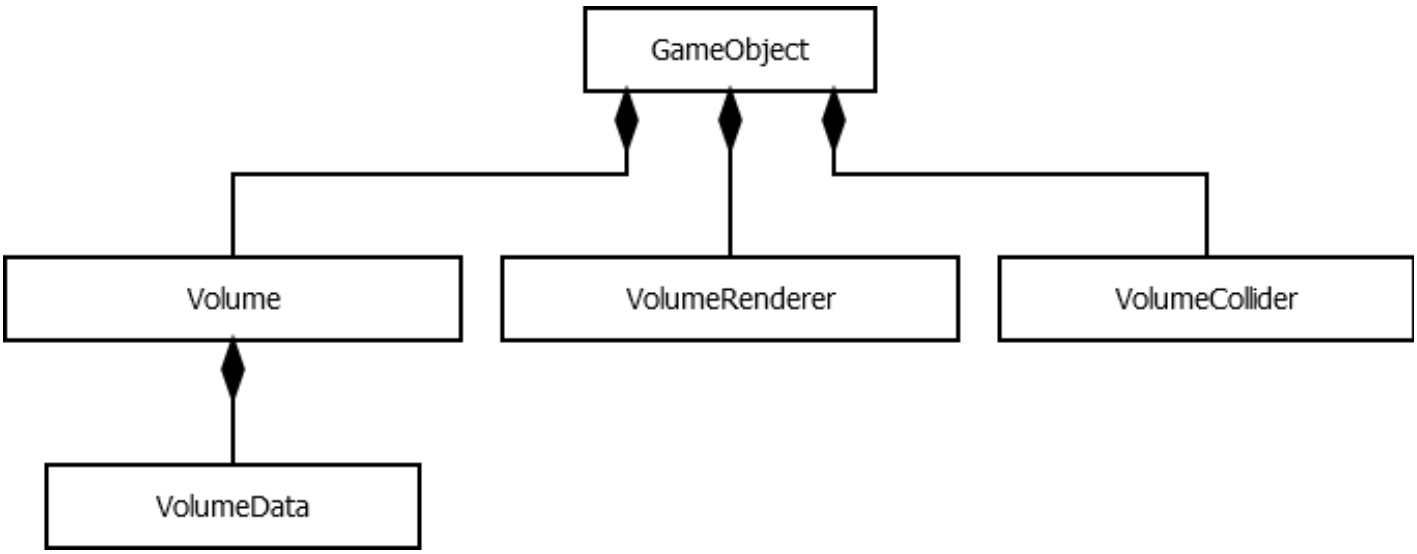
Key Components

We have tried to keep the design of Cubiquity for Unity3D as consistent as possible with the approaches used elsewhere in Unity. To this end we have adopted a compoent-based model in which the user can add a GameObject to a scene, give it a 'Volume' component to make it into a voxel object, and then add VolumeRenderer and VolumeCollider components to control it's behaviour. With this in mind, the structure of a typical volume is as follows:



The structure of a typical volume

Note that the components shown are actually the base classes and in practice are never used directly. Instead you will use a particular set of subclasses (such as TerrainVolume, TerrainVolumeData, TerrainVolumeRenderer and TerrainVolumeCollider) depending on the kind of volume you would like to create. We now look at each of these components in more detail.

The Volume Component

Adding a Volume component to a GameObject makes it into a voxel object. The Volume class does not provide much functionality on its own as most behaviour (rendering, physics, etc) is implemented in the other related components. Instead the class acts more to tie the other components together.

Volume components also have custom inspector implemented which allow you to edit the volume in an intuitive way. For example, the TerrainVolume custom inspector exposes the sculpting and painting tools which can be used to build your own terrain in the Unity3D editor. The inspector also has a 'Settings' panel where you can choose which volume data is being used.

The VolumeRenderer Component

The visual appearance of the volume is controlled by the VolumeRenderer, and if a VolumeRenderer is not present then the volume will not be visible in the scene. The role is similar to that of the MeshRenderer for Unity's Mesh class, and this is no coincidence considering that meshes are used internally by Cubiquity to display the volume data. The VolumeRenderer exposes properties including the material of the volume and its shadowing behaviour.

The VolumeCollider Component

The VolumeCollider component should be attached to a volume if you wish a collision mesh to be generated. This will allow other object in the scene (such as rigid bodies) to collide with the volume. There are currently no properties (the VolumeCollider is simply present or not) but these will likely be added in the future.

The VolumeData Component and Cubiquity Assets

Cubiquity for Unity3D wraps voxel databases with a class called VolumeData, and more specifically with its subclasses called TerrainVolumeData and ColoredCubesVolumeData. These are very thin wrappers, which basically just store the filename of the voxel database and provide functions to get and set the voxel values.

Because VolumeData is derived from Unity's ScriptableObject class it is possible to turn it into a Unity Asset. You can then select the asset in the project view to see the path to its voxel database in the inspector, and you can drop it onto the 'Volume Data' field which is found under 'Settings' in the Volume inspector.