

Voxel Destruction Pro Documentation

Contents

| | |
|---------------------------------|---|
| Setup guide..... | 2 |
| Voxel objects..... | 3 |
| VoxDataProvider..... | 4 |
| Performance optimizations | 5 |
| Features | 6 |
| FAQ | 7 |
| Support | 8 |

Setup guide

1. **Add Voxel Manager:** First you probably want to add a VoxelManager to your scene, this allows you to use the Quick setup button on the Voxel object. VoxelManager also allows you to use Model caching, which will save already read VoxelDatas and reuse it instead of reading the file over and over again. The VoxelManager will use DontDestroyOnLoad and will automatically destroy duplicate instances. You can find a prefab for a default VoxelManager inside the Voxel Destruction Pro base folder.
2. **Import Vox files:** Add the vox file of the models you want to use to the StreamingAssets folder, there it will be included in build and the package can access the files. Make sure to not use any special characters, best practice is to keep everything lower case without spaces and only alphabetic characters.
3. **Create Voxel object:** You can create a new object and one of the Voxel objects to it (The Voxel objects section describes which voxel object you should use). You can then press the “Quick setup” button and if your Voxel manager settings are valid it should automatically create the Mesh Filter, Renderer and collider for you and assign the default settings.
4. **Vox file data provider:** Add a VoxFileDataProvider to allow the Voxel object to read vox files. You will need to set the name of the vox file you imported into Streaming assets into the modelPath field.
5. **Set parameters:** You can now customize the parameters of the Voxelobject, most settings are found inside the ScriptableObjects, like the MeshSettings. You can create your own settings by right clicking in project, going to “VoxelDestruction” and selecting the setting type you want to create. Using ScriptableObjects makes it easy to reuse your settings across multiple voxel objects without having to change the settings on each one.

Voxel objects

The package is divided into multiple voxel objects that can be used for different purposes; you can also override one to add your own features:

- **VoxObjBase:** This is the base class for all Voxel Objects, you can use it to create your own voxel objects. It contains some basic functionalities that can also create a mesh using the greedy mesher, but there are no destruction functionalities, and the voxel object will always be static.
- **IsolatedVoxelObj:** Inherits VoxObjBase, adds the isolation feature to the voxel object that can be used to identify separated parts of a voxel object and removes them.
- **DynamicVoxelObj:** Inherits IsolatedVoxelObj, contains all the logic for destruction. Contains the AddDestruction functions that allow you to destroy voxel objects.

You can create your own Voxelobjects by creating new scripts and inheriting any of these. You can override the CreateJobs method and create the jobs used by your voxelobject in there (just make sure to also call the base method). The DisposeAll method can also be overridden, and you can dispose the jobs there.

VoxDataProvider

In order to make it easier to procedurally create and assign voxel datas, VoxDataProviders are used. These need to be added directly to the game object that has the target voxel object. These VoxDataProviders also come with 2 buttons in the editor to preview the voxel data without having to run the game. Note that a Voxel object is not required to have a VoxDataProvider, since the AssignVoxelData function can also be called from another script. The package comes with 2 VoxDataProviders:

- **VoxFileDataProvider:** This one handles the reading of MagicaVoxels vox files and converts the data into VoxelData.
- **VoxCubeDataProvider:** A simple example on how easy it is to procedurally generate VoxelData. You can input a size and color and it will procedurally generate a cube for you.

To create a custom VoxDataProvider create a new script that inherits from VoxDataProvider and override the Load function. Make sure to call

```
base.Load(editorMode);
```

before running your calculations. The editorMode bool tells you if the function is being run outside of playmode.

Next you can create your voxel data and then call:

```
targetObj.AssignVoxelData(voxelData, editorMode);
```

Performance optimizations

Voxel destruction is a very CPU intensive process, it requires fast mesh creation times, spawning new Fragments at runtime and a lot of other performance intensive operations. Here are some guidelines to optimize the performance of your voxel game:

- **Use IL2CPP and Incremental GC:** The Garbage collector (GC) cleans up the memory, this can result in sudden spikes of CPU usage. Under player settings you can find an option to use Incremental GC, this will spread the cleanup over multiple frames reducing the lag. Additionally, the unity Job system performs much better when using IL2CPP, so if you have the option to use it I strongly recommend you do.
- **Configure the settings in the right way:** The package contains some settings that allow you to customize the voxel destruction to your needs, these can also be used to improve the performance. All parameters have tooltips that describe what they are used for.
- **Split large objects into smaller ones:** Obviously larger voxel objects will take up more performance, so whenever you can try to split them into smaller pieces. This may affect some features, like isolation.
- **Use simpler colliders:** The package supports Mesh colliders and Box colliders, if you want better performance you can try to use Box colliders wherever it is possible. You can for example make the fragments use Box colliders instead of Mesh colliders.

Features

The main use case of this package is to create games that require some type of Voxel destruction. Whether you only want some objects in your game to be destructible or create a casual mobile game that requires voxel destruction, this package allows you to do so. However, you can also use the components of this package for other functionalities:

- **Greedy Meshing:** Greedy meshing can be used in several applications, for example if you want to create a simple Minecraft clone you probably want to implement greedy meshing. The greedy meshers of this package use the IVoxMesher interface (You can read about the functionalities inside the interface).
- **Connected component labeling:** CCL is used to find isolated pieces inside Voxel objects. It takes in an array of bytes defining the voxel states and an int3 length, it then returns an array of ints that defines the label of each voxel. For inactive voxels the label is 0 else the label describes a piece inside the voxel object. These labels are not in numeric order, but every piece that is not connected will have its own label.
- **Destructors:** These jobs are responsible for identifying the voxels that are effected by a destruction, there are multiple implementations that use different ways to identify the target voxels.
- **Fragmenters:** These jobs create the voxel fragments from the destroyed voxels, there are multiple implementations and also the option to not use any fragmenters for no fragments.

FAQ

How can I further improve the performance of this package?

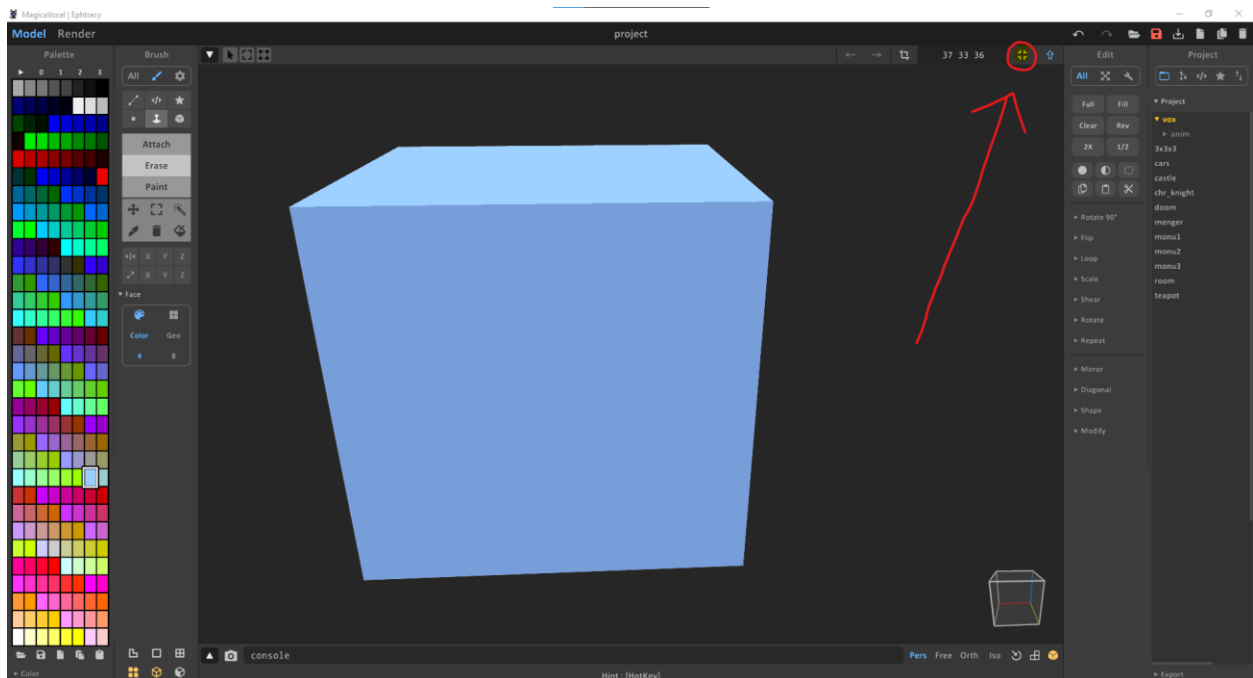
Take a look at the Performance optimizations chapter. If you are looking for even more performance and smaller voxel sizes you can look into Voxel raytracing, a technique used by many voxel engines. Unity is also working on DOTS (Data oriented technology stack), which could strongly improve the performance of this package.

Can I animate voxel objects?

It is not possible to use something like a skinned mesh renderer with these voxel objects, but you can always split objects into multiple parts and animate them.

My Voxel object keeps disappearing or turns into a fragment, why?

If you are using Isolation on your voxel objects, it could be the case that the bounding box of your object is too large. The isolation origin describes what side of the object is connected to a solid surface and is the basis for isolation, but if your bounding box is a bit larger than the object itself it could be the case that no voxel actually touches the basis. To fix this you need to make the bounding box smaller inside MagicaVoxel, there is a button that will automatically do this for you.



Support

For further questions you can always reach out to me at atangameshelp@gmail.com, just make sure to read FAQ first, which covers a lot of basic problems that could occur.