

PICCA VOXEL



The *Tiny* Voxel Toolset for Unity

Contents

| | |
|--|----|
| Get in Touch..... | 2 |
| Version History..... | 3 |
| What is PicaVoxel?..... | 4 |
| What is PicaVoxel <i>Not</i> ?..... | 4 |
| PicaVoxel Features..... | 4 |
| Getting Started..... | 5 |
| PicaVoxel Volumes..... | 6 |
| The Volume Inspector..... | 7 |
| Editing | 9 |
| Animation..... | 10 |
| Voxel Values..... | 12 |
| Runtime-Only Mesh | 12 |
| Mesh-Only Copy..... | 12 |
| Mesh Assets and Storage (Version 1.2.5 Onward) | 13 |
| Multi-part Volumes..... | 14 |
| Importing | 14 |
| Particles..... | 15 |
| Utility Scripts | 16 |
| Scripting | 18 |

Get in Touch

<http://picavoxel.com>

[@garethiw on Twitter](https://twitter.com/garethiw)

support@picavoxel.com

Version History

05 May 2015: 1.2.5

- FIX: Having a volume with a chunk with no voxels in it could possibly cause a crash on some OpenGL platforms.
- FIX/FEATURE: PicaVoxel now creates Mesh assets for all chunks on all animation frames on all volumes. This means that you can now add PicaVoxel objects to prefabs correctly, as either full Volumes or Mesh-Only Copies.
- FEATURE: Mesh Compression setting exposed in Volume inspector.
- FEATURE: Magica/Qubicle import will use filename (minus extension) as GameObject name if the name isn't changed in the import window.
- FEATURE: Updated manual with changes from 1.0.0 to 1.2.5.

16 Apr 2015: 1.2.1

- FIX: UnityEditor reference in Chunk.cs caused error on standalone build. Encase in #if UNITY_EDITOR to fix in the meantime.

20 Mar 2015: 1.2.0

- FIX: Unity 5 API upgrades. All scripts now Unity 5 compatible.
- FIX: Prefab connection breaks if PicaVoxel volume used as part of a prefab.
- FIX: Chunk rebuild loses layer (Chunk layer is now set to parent volume's layer)
- FIX: Clicks pass through voxel editor UI.
- FIX: Meshes generated on awake even if Runtime-Only Mesh is disabled.
- FEATURE: Multi-part Volume inspector tools
- FEATURE: Move animation frames forward/backward.
- FEATURE: Animation frame copying (helpful for importing multiple frames from Magica/QBC)
- FEATURE: Select box tools (Fill, Clear, Nudge)
- FEATURE: Allow separate mesh generation methods for collider/render (currently edit-time only)
- Lots of minor scripting enhancements and optimisations.

18 Feb 2015: 1.1.0

- FIX: Error when turning off Runtime-Only Mesh
- FIX: Intermittent gizmos error when using "Draw Grid" and "Draw Wireframe" toggles
- FIX: Not compatible with Windows Store platform.
- FEATURE: Meshing Mode toggle – Culled or Greedy (default Greedy). Implemented because under some circumstances Culled meshing is faster than Greedy, which can help when updating frequently during runtime. This of course at the cost of extra triangles.
- FEATURE: Constructor and Destructor utility scripts. See new ConstructorDestructor demo for example. Tutorial video and manual entries to follow.
- FEATURE: Brush painting "Replace" toggle. Normal brush painting behaviour is to attach to faces (same as adding single voxels). Replace toggle lets you paint the brush at its anchor over an existing voxel.
- FEATURE: All Frames editing propagation. Changes made to one animation frame will be made to all frames whilst this option is toggled. Includes single + box voxel add, subtract and paint as well as brush add, subtract and paint.

09 Feb 2015: 1.0.0 (First Release)

- Original release

What is PicaVoxel?

PicaVoxel is two things:

1. An easy-to-use voxel editing toolset that allows you to create and manipulate vertex-colored voxel models directly inside the Unity editor.
2. A mini game engine that allows you to use the voxel data in your scene at runtime.

What is PicaVoxel *Not*?

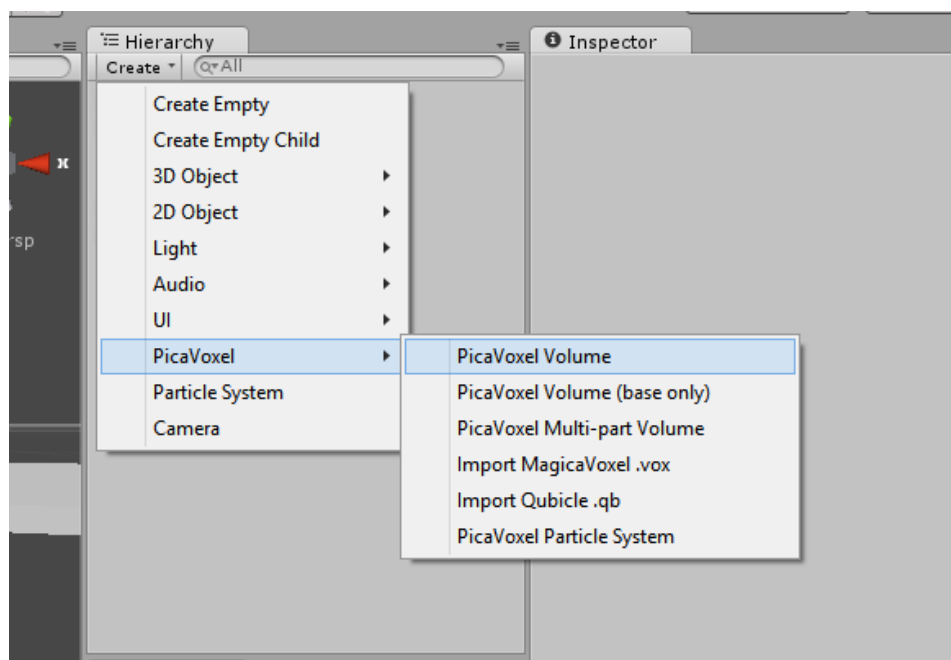
PicaVoxel is not an engine with which to create a Minecraft clone. It does not support infinite, disk-streamed worlds. Neither does it support texturing of individual voxels.

PicaVoxel Features

- In-scene voxel volume editor.
- Voxel data available at runtime.
- Frame-based animation. Like creating a 2D animated spritesheet, but with 3D voxels!
- Position and rotate your voxel volumes freely in your scene.
- Optimised mesh generation.
- Create a basic model from the voxel volume (meshes without voxel data).
- Runtime-only mesh option - reduce scene disk size by not saving meshes with the scene and having them generate when the scene is loaded.
- [MagicaVoxel](#) import (.vox format).
- [Qubicle](#) import (.qb)
- Multi-part volumes for large objects.
- Unity physics supported - convex or concave MeshColliders.
- Manual collision detection at runtime - detect a voxel at an arbitrary Vector3 world position - even when the volume is rotated.
- Utility scripts included to help with manual collision detection, voxel destruction, and animation.
- Voxel particle engine - wraps Unity Shuriken particle system. Explode/destroy your voxel volumes into particles!
- Five sample games with fully commented source - helping you get the most out of PicaVoxel.

Getting Started

Using PicaVoxel starts with the new PicaVoxel submenu in the create menu:



PicaVoxel Volume

A standard 32x32x32 voxel volume with all voxels filled in.

PicaVoxel Volume (base only)

A 32x32x32 volume with the "bottom" (Y=0) filled in.

PicaVoxel Multi-part Volume

For when you need a large (larger than 64*64*64) volume. A window will appear with options for creating the multi-part volume. See [Multi-part Volumes](#).

Import MagicaVoxel/Qubicle

Imports a MagicaVoxel .vox or Qubicle .qb file and creates a volume based on the contents. See [Importing](#).

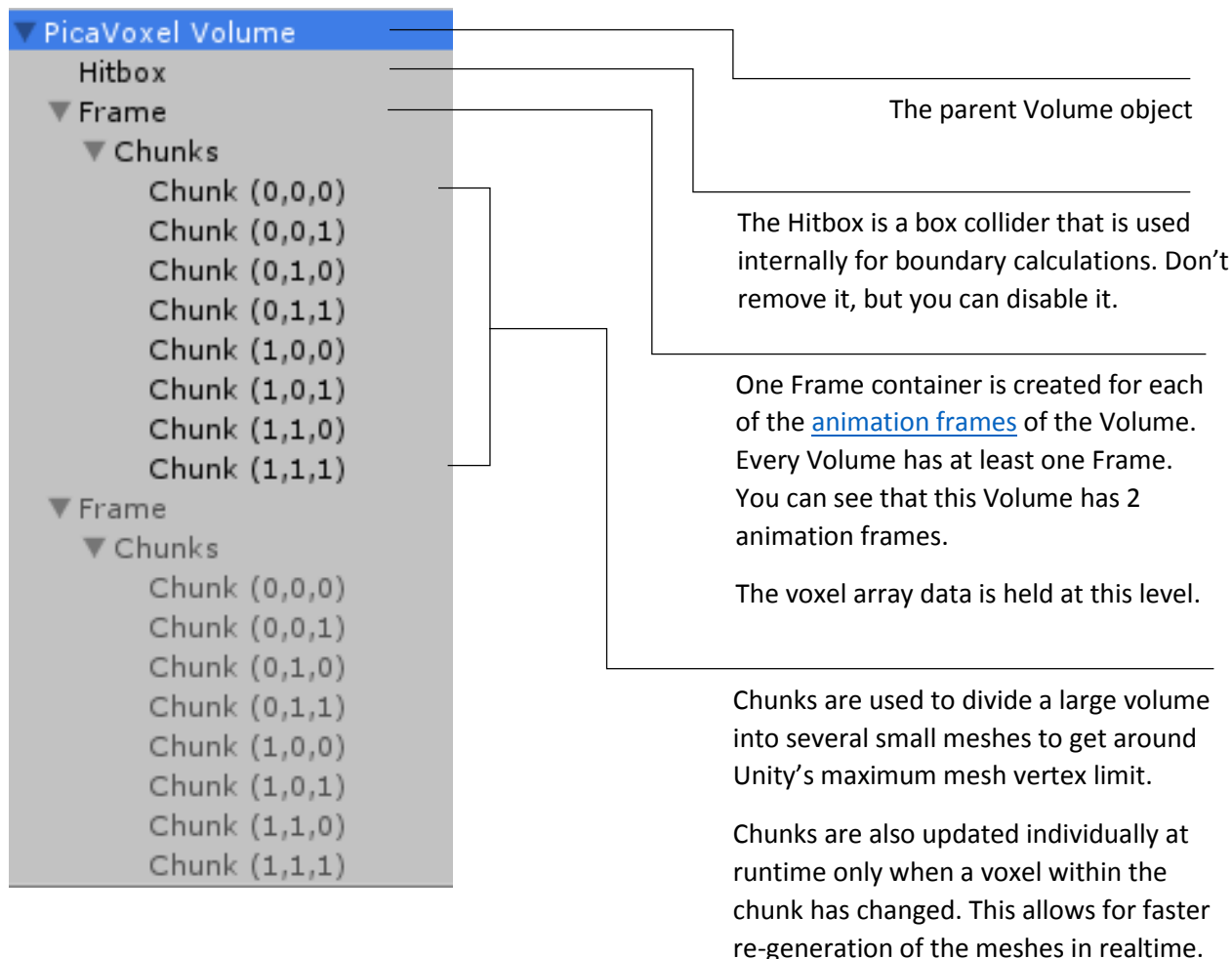
PicaVoxel Particle System

Activates the automatic particle functionality of PicaVoxel. Only one PicaVoxel Particle System may be added to a scene at a time. See [Particles](#).

PicaVoxel Volumes

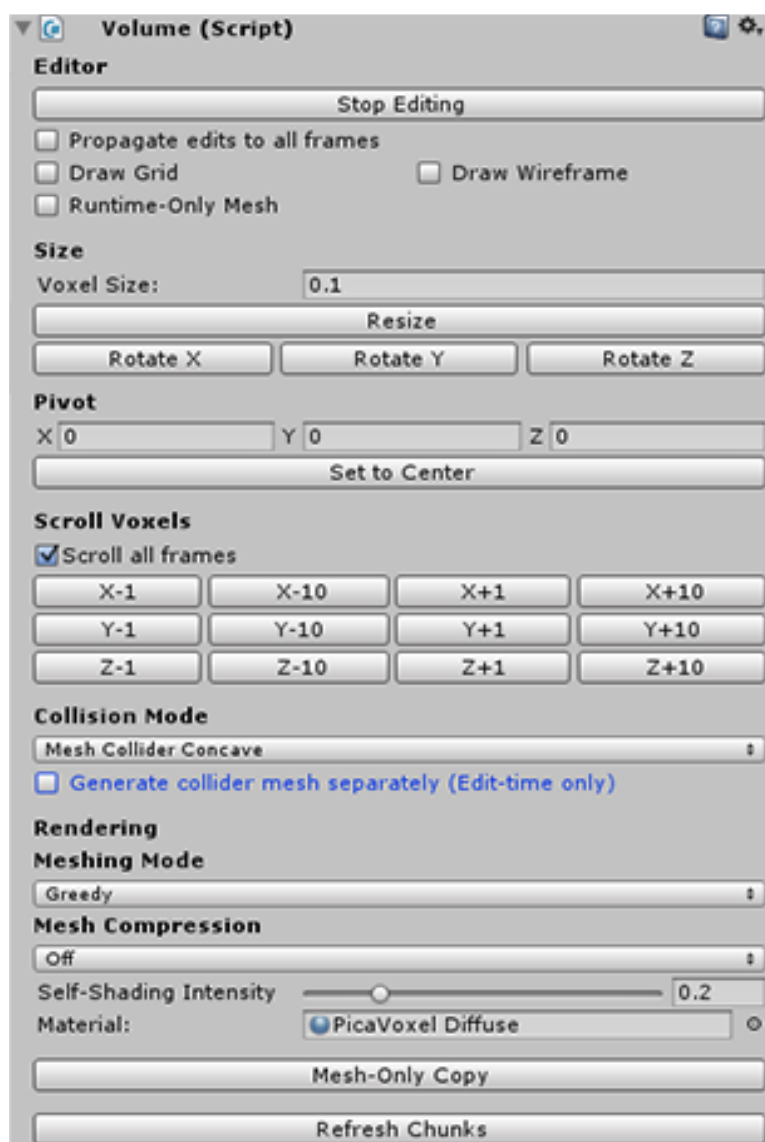
The Volume is the heart of PicaVoxel. Essentially, it holds all of the voxel data in an array and deals with the generation of the mesh to represent that data; though there is a little more complication to that simple overview.

It's important to understand the hierarchy of the Volume object, so let's take a look:



The Volume Inspector

Select a PicaVoxel Volume in the scene hierarchy to bring up the inspector for the Volume.



- **Start/Stop Editing:** Activates and deactivates the voxel editing tools for this Volume. See [Editing](#).
- **Propagate edits to all frames:** Changes made in one animation frame will be made to all other frames.
- **Draw Grid:** While editing, overlays a faint grid to show the extents of the Volume.
- **Draw Wireframe:** While editing, enables/disables drawing of the mesh wireframe for this Volume.
- **Runtime-Only Mesh:** See [Runtime-Only Mesh](#).
- **Size:** The cube size, in units, of the individual voxels in this Volume. **Change this value to resize the Volume instead of using the Transform scale.**
- **Rotate X, Y, Z:** Rotate the voxels within the Volume by 90 degrees around the specified axis. Note that this is rotating the voxels within the voxel array and is not the same as rotating the Transform.
- **Pivot:** Set the X, Y, and Z position offset of this Volume. 0, 0, 0 represents the left, bottom, front of the Volume.
- **Scroll Voxels:** Nudges all of the voxels in the Volume in the specified direction by the specified amount. You can scroll by 1 or 10 voxels negatively or positively along each axis. The voxels wrap around when they reach the edge of the Volume.
- **Collision Mode:** Determines whether collision meshes are generated for each of the chunks in this Volume. You should only use collision meshes when combining with Unity's physics system. Use Concave meshes for static objects (scenery), and Convex for moving Volumes, along with a Rigidbody (which should be added on the top-level Volume object).
- **Generate mesh collider separately:** If you wish, you may choose to use (for example) a Culled mesh for rendering, and a Greedy mesh for collisions. Only use in very specific cases.
- **Meshing Mode:** Greedy (default) or Culled. Greedy meshing uses an algorithm to break down the generated mesh into larger areas, using less triangles for rendering. Culled rendering uses one quad per exposed voxel face. Greedy meshing is slower than Culled, so for Volumes that are going to be procedurally generated or destroyed in runtime you may find Culled is a better option.
- **Mesh Compression:** You can turn on mesh compression to save disk space (scene file size as well as stored mesh sizes) in your project at the expense of mesh quality. Higher compression adversely affects colorization and vertex positioning.

- Self-Shading Intensity: PicaVoxel adds a 1-voxel shadow where appropriate on the mesh. This value determines how dark the shadow is. A value of 0 will skip the shadow calculation entirely.
- Material: Set a material that will be propagated to all of the chunk meshes on this Volume. If you wish to, for example, add a tint color to the Volume you should create a new material using the PicaVoxel Diffuse shader, set the tint color and then use the new material here.
- Mesh-Only Copy: See [Mesh-Only Copy](#)
- Refresh Chunks: Re-generates all of the meshes on all of the chunks across all of the animation frames in the volume. Useful if something goes wrong or if the stored mesh assets are deleted.

Editing

When you click “Start Editing” on a Volume, you are presented with the editing toolbar in the scene view:

Color/Value toggle, value slider and input: Use the dropdown to toggle between Color and Value editing modes. See [Voxel Values](#) for more details.

Use the slider or textbox to change the value to paint with.

Color palette: Each Volume has its own stored palette of 25 colors, but you are not limited to just those 25 colors to paint with.

Use the color field to change the currently selected color.

Attach, Subtract and Paint individual voxels: Single-click or click+drag to add, remove and paint individual voxels. Note that you can only attach new voxels to existing ones. Use paint to change the color/value of

Box Attach, Subtract and Paint: Click+drag to add, remove and paint voxels in the dragged area.

Select Box: Define an area to create a brush from. Drag the handles in the scene view to create a selection of voxels, and then click the “Create Brush” button that appears below the edit tools.

Brush Attach, Subtract and Paint: Only available when a brush has been created with the Box Select and Create Brush tools.

Subtracting a brush shape from the Volume will remove all voxels in the brush at the brush’s position in the Volume.

Likewise, painting will replace the colors and values of voxels in the Volume with those from the brush. Only active voxels in the Volume are affected.

Color and Value Pickers: Sets the current color or value to that of a left-clicked voxel that is part of the Volume.

Note that if one of the palette colors is the exact color of the chosen voxel, that palette color will become active.

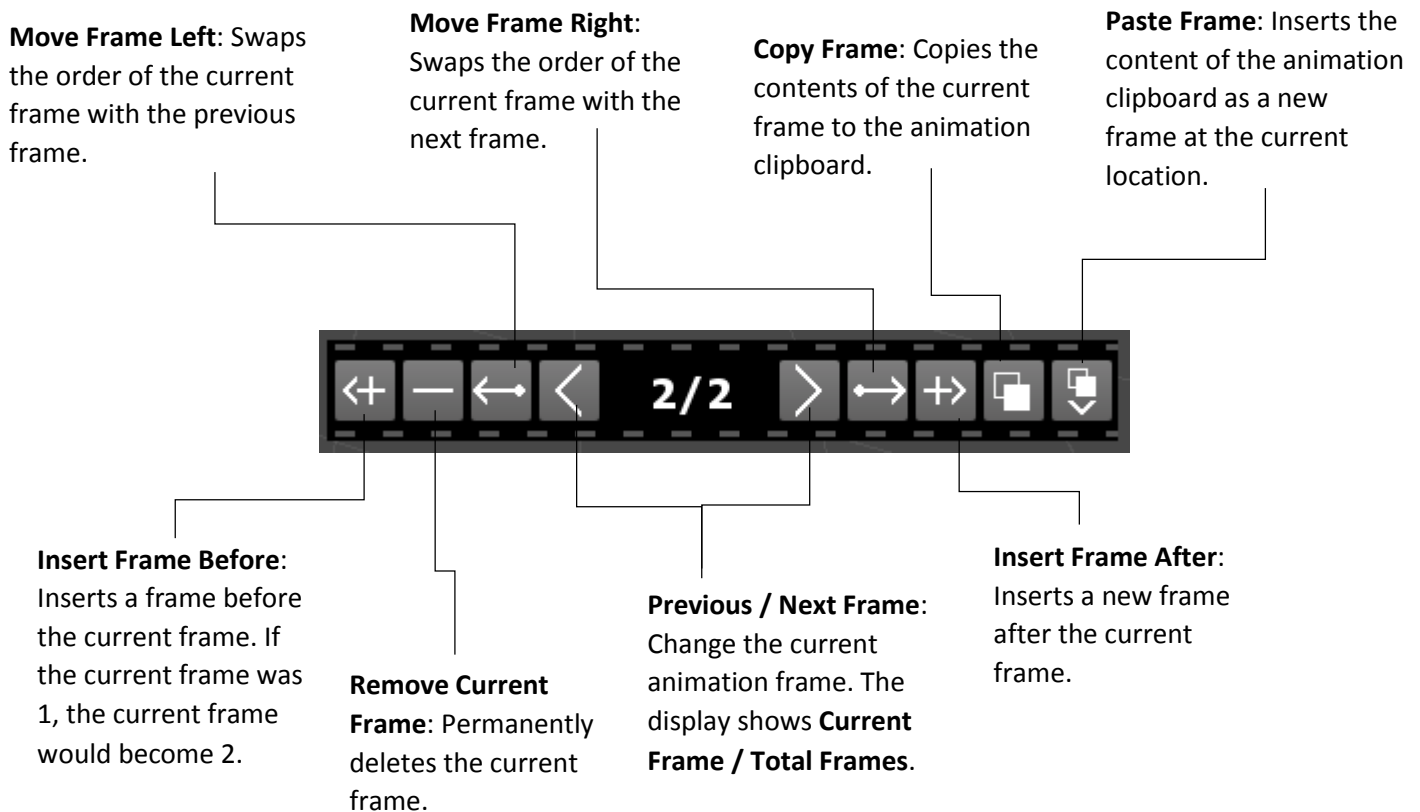
If no matching color is found in the palette, the currently selected color will be changed to match the chosen voxel.



Animation

Animation in PicaVoxel is a simple concept: One Volume is made up of one or more animation Frames. Each Frame holds the voxel data for the frame along with the chunk meshes for that Frame. Animation works by simply activating/deactivating the Frame objects at runtime.

When a new Volume is created, it has one animation frame. Add, remove, and switch between frames using the Animation toolbar which is displayed in the lower-right of the Scene view when editing a Volume:



When adding frames, the current frame's contents are duplicated, making it easier to create an animation sequence.

To play the animation at runtime, you can use the included Basic Animator script, or change the CurrentFrame field on the Volume manually.

Creating an animation from multiple MagicaVoxel or Qubicle imports

PicaVoxel doesn't currently have any functionality for creating an animation from multiple imports, but the same thing can be achieved with the following workflow:

1. Import all of the files into Unity using Create->Qubicle/Magica Import
2. Use the first animation frame volume as your "master" volume. You will add all the other frames to this. Click Start Editing on this volume.
3. Select the second animation frame volume and click "Start Editing".
4. Click the Copy Frame button (second from right on the Animation toolbar)
5. Select the "master" volume from step 3. Click the Paste Animation frame button (last button on the right of the Animation toolbar)

6. Repeat steps 3-5 for each animation frame.
7. Click “Stop Editing” on the "master" volume.
8. Delete all the other individual frame volumes.

Voxel Values

As well as a Color and an Active status, every individual voxel in a volume also has a Value. The Value is a byte (0-255), and defaults to 128. You can use the Value painting mode in the volume editing tools to change the value of voxels.

How you use the value is up to you. The most obvious usage is in determining which voxels can be destroyed. You could also use it to designate voxels as a certain “type” in your game (i.e. water, glass etc.) or as a mask for detecting collisions.

The included Exploder utility script (see [Utility Scripts](#)) has a Value operation to determine which voxels are destroyed when its Explode() method is called.

Runtime-Only Mesh

Although the meshes generated by PicaVoxel are optimised, they can still take a large amount of disk space in your scene. Because of this, you can designate large voxel volumes as Runtime-Only Mesh volumes.

Once the Runtime-Only Mesh option is ticked, the volume’s mesh will no longer be visible in the scene at edit-time, and you will no longer be able to edit the voxels in the volume. The meshes are destroyed and scene space disk and memory usage will be decreased.

At runtime, the meshes will be automatically generated as soon as the volume’s Awake() method is called. Usually, this occurs when the scene is loaded. You will then see the voxel volumes exactly as they were before you turned on Runtime-Only.

If you need to make changes to your volume, simply toggle Runtime-Only Mesh off again, and the meshes will be re-created in the editor and the Start Editing button will be available.

Mesh-Only Copy

If you don’t need access to the voxel data for a volume in your game, use the Mesh-Only Copy button to create a duplicate of the volume without the PicaVoxel scripts attached. The object hierarchy is preserved, so you can still use the Basic Animator to play the animation if you need to.

Note that there’s no way to get back from a mesh-only copy to an editable PicaVoxel Volume, so only use this once you’ve finalised your volume editing, or keep a copy of the original volume on a separate scene or in a prefab.

Mesh Assets and Storage (Version 1.2.5 Onward)

As of version 1.2.5, PicaVoxel now creates physical Mesh assets in the Unity asset database for each chunk on each animation frame on each Volume. This is necessary to support adding PicaVoxel volumes to prefabs without having to re-generate the meshes from voxel data upon instantiation.

The meshes are stored in the PicaVoxel folder within your Unity project, in a new folder named “Mesh Storage”. This folder will be created the first time a Volume is created or re-drawn. Each PicaVoxel volume that you create in your project is assigned an ID (GUID as a string), and a new folder is created inside the Mesh Storage folder for each volume, with the ID as the volume name.

All meshes, including collider meshes, are saved as assets in this folder structure. It is recommended that you do not delete, rename or otherwise change anything inside this folder – but if you do, PicaVoxel will re-create the folder and mesh assets when the Volume is re-generated.

The stored meshes are used differently depending on the Volume:

Standard PicaVoxel Volumes, or Mesh-Only Copies

Every Volume has its own set of meshes created in the Mesh Storage folder. If you make a copy (via copy/paste or Duplicate) of the Volume, the copy will also have its own set of meshes created with it. This means that you can edit duplicated Volumes and the changes will not reflect on the original’s meshes.

When you use the “Mesh-Only Copy” function to create a Volume without realtime voxel data, the copy will also have its own mesh assets created, thus keeping it separate from the original Volume.

Prefabs

When you create a Prefab from a PicaVoxel Volume, the prefab **does not** get its own mesh assets. This means that all instances of the prefab will share a mesh at edit-time and changes made to one instance will reflect on all of the others.

Runtime

If any changes are made to a Volume at runtime (by way of explosions, SetVoxel(), CreateChunks() etc.), new meshes will be assigned to that Volume and will be used for the duration of that Volume’s lifetime. This means that instantiated prefabs can be changed individually without the change reflecting on all other instances of that Volume.

Multi-part Volumes

PicaVoxel restricts the maximum size of a single Volume to 64*64*64 (262,144) voxels for performance reasons. If you need to create a volume larger than this, you can use the Multi-Part Volume creator to make a larger object that is made up of several individual Volumes arranged in a grid.

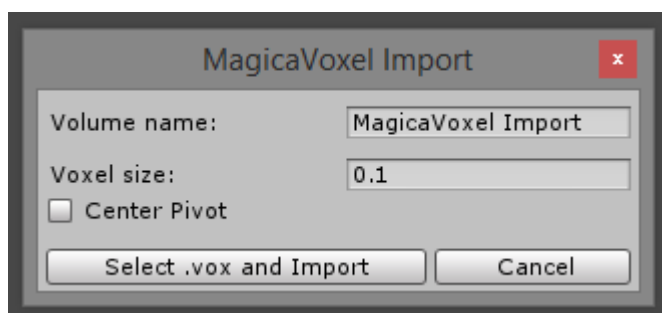
As of Version 1.2.0, an Inspector for Multi-Part volumes will now appear when you select the root GameObject of a Multi-Part Volume in the hierarchy. The Multi-Part volume inspector contains a subset of the inspector tools available on a standard PicaVoxel Volume.

Importing

PicaVoxel can import MagicaVoxel .vox and Qubicle .qb formats. Note that Quible .qbcl (project) formats are not compatible, and therefore a licenced version of Qubicle is required.

MagicaVoxel Import

To import from MagicaVoxel, choose “Import MagicaVoxel .vox” from the PicaVoxel Create submenu. A new window will appear:

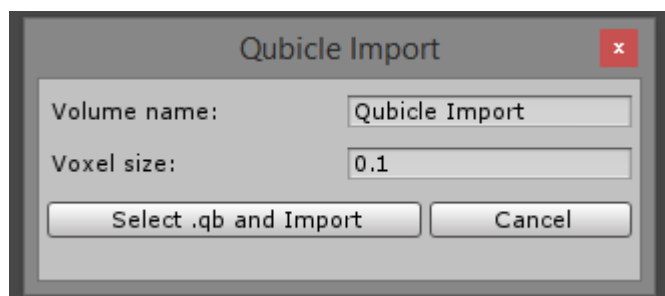


- Volume Name: The name of the Volume that will be created.
- Voxel size: The initial size of the individual voxels.
- Center Pivot: Set the pivot to the centre of the object after creation.

If the .vox creation exceeds 64 voxels in any dimension, the .vox will be imported as a Multi-Part Volume. It's important in this case to ensure the name, voxel size and pivot are set as required as it is not easy to alter them after importing a Multi-part Volume.

Qubicle Import

To import from Qubicle, choose “Import Qubicle .qb” from the PicaVoxel Create submenu. A new window will appear:



- Volume Name: The name of the Volume that will be created.
- Voxel size: The initial size of the individual voxels.

Qubicle creations are made up of one or more “Matrices”, which are treated as individual volumes when imported into PicaVoxel. All of the matrix volumes are placed inside a parent object (named as the chosen Volume Name). The matrix volumes will retain the name they were given in Qubicle. Because the matrices all have their own position relative to the centre of the creation, the pivot will always be the centre as set in Qubicle.

As with the .vox import, any individual matrices in the .qb that exceed 64 voxels in any dimension will become a Multi-Part Volume.

Note: As of Version 1.2.5, if you do not change the default volume name on a Qubicle or Magica import, the volume (or root GameObject) will be named the same as the imported file (minus extension).

Particles

The PicaVoxel Particle System is a wrapper around the standard Unity Shuriken particle system. When you add a PicaVoxel Particle System to the scene, a Shuriken system is added that has been set up with some defaults, such as rendering with cube meshes and using the PicaVoxel Diffuse shader. Feel free to change any of the additional options on the Shuriken system to suit your needs.

The actual PicaVoxel Particle System script is a singleton, allowing only one of them to be placed in the scene. It is referenced directly by the Exploder utility script, and by the Volume script. This allows you to call Explode() on the exploder, and Destruct() on a Volume to spawn explosion particles without any extra work.

There are a number of exposed options on the PicaVoxel Particle System:

| | |
|---------------------|-------------------------------------|
| Particle Lifetime | 5 |
| Max Batch Particles | 100 |
| System | None (Particle System) |
| Collide Positive X | <input checked="" type="checkbox"/> |
| Collide Negative X | <input checked="" type="checkbox"/> |
| Collide Positive Y | <input checked="" type="checkbox"/> |
| Collide Negative Y | <input checked="" type="checkbox"/> |
| Collide Positive Z | <input checked="" type="checkbox"/> |
| Collide Negative Z | <input checked="" type="checkbox"/> |
| Bounce Multiplier | 0.5 |

- Particle Lifetime: The initial lifetime of particles spawned with the Explode and Destruct calls.
- Max Batch Particles: The maximum number of particles per volume that can be spawned with a single Explode or Destruct call. Best kept to between 50 and 300 particles.
- Collide direction toggles: By default, the Shuriken particles are set up with

standard Unity physics collisions. If you are *not* using Unity physics and mesh colliders, you can turn these custom collision toggles on to have the particles collide with active voxels in the scene in each direction. This can be a huge hit on performance, depending on the number of particles and the number of active Volumes in the scene. Experiment, and use with care!

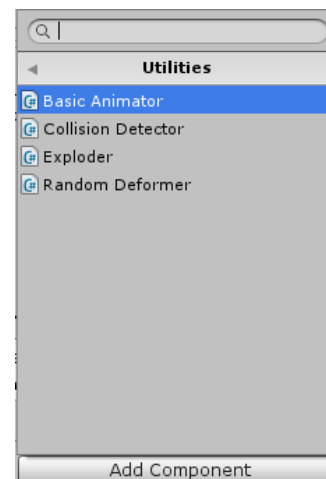
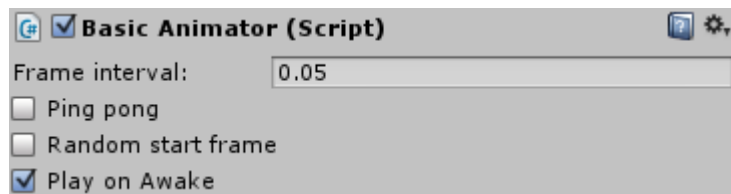
- Bounce Multiplier: When using voxel collision, the particle velocity is multiplied by this amount upon collision.

Utility Scripts

There are six utility scripts included with PicaVoxel to give you a headstart with using voxels in your game. You can find the scripts in Add Component -> PicaVoxel -> Utilities.

Basic Animator

Must be attached to a PicaVoxel Volume at parent level.



Use the basic animator to animate through the animation frames on your Volumes.

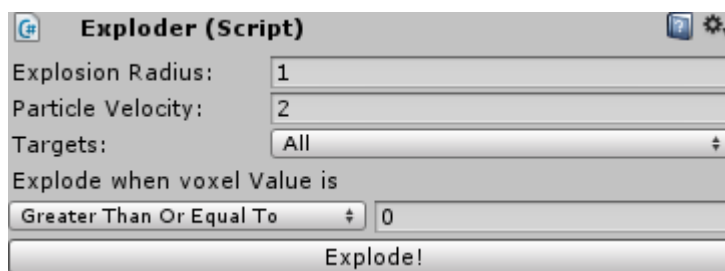
- **Frame Interval:** The time (in seconds) between frames of animation.
- **Ping Pong:** When ticked, the animation will play forwards and then backwards before looping. If not ticked, the animation will play from start to end, then loop directly back to the start.
- **Random start frame:** Usually Frame 1 will be the start frame, tick this to choose a random start frame. Useful when you have many copies of the same object and want to stagger their animations.
- **Play on Awake:** Start playing the animation as soon as the object awakes at runtime.

You can manually call the following methods on the Basic Animator:

- **Play():** Starts playing the animation from the current frame.
- **Pause():** Stops playing the animation but does not reset the frame number.
- **Reset():** Stops playing and resets to frame 1 (or a random frame if Random start frame is set).

Exploder

The Exploder script can be attached to any Unity object. When the Explode() method is called, all voxels within the specified radius are tested and destroyed according to the parameters set on the Exploder. The voxels can be a part of any Volume, rotated in any way.



- **Explosion Radius:** The radius (in units) from the world transform position of the Exploder in which voxels may be destroyed.
- **Particle Velocity:** If a PicaVoxel Particle System is in the scene, particles will be created for the explosion. This is the velocity of the particles created. Note that this is multiplied by the normalized direction from the Exploder to the voxel, so particles will always explode outwards.
- **Targets:** All, All But Self, or Self Only. If the Exploder is placed on a Volume, use the Targets setting to determine if the voxels in the Exploder's volume are also destroyed.

- Value test function: Only explode voxels with a value that matches the set criteria. Use when you designate parts of a volume with values that denote their destructibility. For example, painting glass with a value of 10 and the surrounding wall with a value of 128 and setting the exploder to explode when the value is Less Than 128 would result in the glass voxels being destroyed, but not the wall.
- Explode!: Use this button in runtime to test explosions. Can also be used at edit-time, but be careful not to edit the voxel volume afterward, as this will save the exploded voxels.

Collision Detector

Attach a Collision Detector to any object that you need to test for collisions with voxels.

Set one or more Detection Points on the Collision Detector. These are Vector3s representing distance (in units) from the transform position of the Collision Detector. At runtime, each of the defined points are tested to see if there is an active voxel at their position. The voxel can be part of any Volume, and rotated in any way.

The Collision Detector has an OnVoxelCollision event that will be called when a collision is detected. Subscribe to the event from your script:

```
GetComponent<CollisionDetector>().Collided += Chopper_Collided;
```

The callback method looks as follows:

```
private void ShmupChopper_Collided(Volume collisionObject, Voxel voxel, Vector3 worldPosition)
{
}
```

- collisionObject is the Volume that the collided voxel is part of. You can test collisionObject.name (for example) to see if the volume is one that you wish to collide with.
- voxel is the actual individual voxel that was collided with. You can test this for value if required.
- worldPosition is the Vector3 position in the world where the collision took place.

Constructor

Must be attached to a PicaVoxel Volume at parent level. See the ConstructorDestructor demo for an example of how to use the Constructor script.

Destructor

Must be attached to a PicaVoxel Volume at parent level. See the ConstructorDestructor demo for an example of how to use the Destructor script.

Random Deformer

Must be attached to a PicaVoxel Volume at parent level.

A simple test script that removes a random number of voxels from the volume at the set interval. The script will continue to remove voxels all the time it is active. Optionally, you can toggle the script to add voxels (of a random color) instead of removing them.

Scripting

PicaVoxel does assume an intermediate level of C# scripting knowledge, outside of simply creating voxel models to use in your own games.

The absolute best way to get to grips with PicaVoxel scripting is to check out the five included sample games:

- **PicaDerby:** A destruction-derby vehicle game, which uses PicaVoxel volumes as the car bodies, with full destructibility. This demo combines PicaVoxel volumes with Unity physics, mesh colliders, and the Exploder utility script.
- **PicaInvaders:** A space invaders demo that uses PicaVoxel collision detection via the Collision Detector utility, as well as the Basic Animator utility, the Exploder utility, and Volume.Destruct().
- **PicaIslands:** A first-person demo combining Unity physics, mesh collision, the Exploder utility, adding voxels to volumes, and moving volumes.
- **PicaRPG:** An example of using larger volumes combined with manual per-voxel collision detection to move a character around a RPG-style landscape. Also uses the Basic Animator utility on the main character.
- **PicaShmup:** A high-end demo that goes all out with voxel destruction, procedural generation of voxel volumes, lots of particles and collision detection.

Aside from the utility scripts, there isn't too much else to learn. The Volume and Frame scripts are the only points of entry you need, and unless you need direct access to the voxel array only the Volume script is worth paying attention to.

Here's a rundown of the useful methods on the Volume script:

Voxel? GetVoxelAtWorldPosition(**Vector3** pos)

Returns any voxel in this volume's current animation frame at a given world position. If the supplied position is not inside the volume, null is returned.

SetVoxelAtWorldPosition(**Vector3** pos, **Voxel** vox)

Set a voxel in this volume's current animation frame at a given world position. Has no effect if the supplied position is not inside the volume.

Frame GetCurrentFrame()

Returns the current animation frame. Useful if you wish to access the voxel array (Frame.Voxels) directly.

NextFrame()

PrevFrame()

SetFrame(**int** frame)

If you want to have direct control over the animation at runtime, these are the methods for you! NextFrame and PrevFrame will wrap around at the extents of the animation. SetFrame will do nothing if the supplied frame is outside of the animation extents. **Note that frame numbers in script are zero-based, unlike the frame numbers shown in the editor.**

Accessing the voxel array directly

If you're planning on procedural generation of voxel volumes, you may need to set the voxels directly in the voxel array. Note that after doing so, you'll need to call `Frame.UpdateAllChunks()` (unthreaded) or `Frame.UpdateAllChunksNextFrame()` (threaded) to redraw the meshes for the volume.

Remember that each animation frame has its own voxel data array, so you'll first need to get the right frame. It's not often you'll be procedurally generating animations, so frame 0 will be the only frame in the volume. Get it with `Volume.GetCurrentFrame()` or `Volume.Frames[0]`.

Once you have a reference to the frame, the voxel data is a one-dimensional array named `Voxels`. For example: `Volume.Frames[0].Voxels[]`.

The calculation for the X,Y,Z position of a voxel in the volume is as follows:

$$X + \text{Volume X Size} * (Y + \text{Volume Y Size} * Z)$$

An example, for setting the voxel at location 10,10,10 in a script attached to a PicaVoxel Volume:

```
GetComponent<Volume>().Frames[0].Voxels[10 + XSize * (10 + YSize * z)] = new Voxel()
{
    Active = true,
    Color = Color.red,
    Value = 128
};
```

Note that `XSize`, `YSize` and `ZSize` are available on both the `Volume` and `Frame` scripts.