# UnityVolumeRendering documentation

## General documentation

- Importing datasets
- Using the imported datasets
- Changing the appearance
- Shadow volumes
- Transfer Functions
- Cross section tools
- Slice renderer
- Editor import settings

## Scripting documentation

- Importing datasets from code
- Raycasting the geometry on CPU

## Other

- SimpleITK integration (for JPEG2000-compressed DICOM and more)

# Importing datasets in the editor

**Table of contents:**



For documentation on how to import datasets from code, see Importing datasets form code.

There are several ways to import datasets into the editor: 1. Through “Volume Rendering” -> “Load dataset” in the menu bar. 2. By drag-and-drop dataset into the “Assets” directory (only RAW/NRRD/NIFTI, Unity 2020.2+) 3. Right click in “Assets” directory and choose “Volume Rendering” -> “Import dataset”

## Importing datasets through the “Volume Rendering” menu bar option

Click “Volume Rendering” in the menu bar and then “Load dataset” and select which dataset you want to import

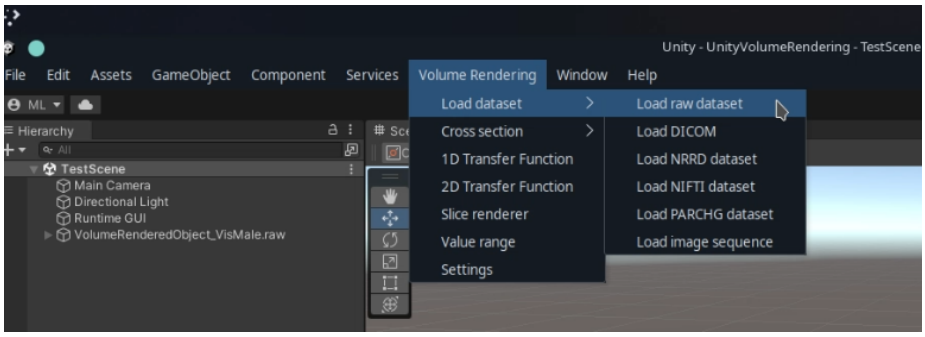


Figure 1:

### Raw datasets

For raw datasets you simply select a single file to import.

An import settings menu will then pop up Here you can set the import setting for the raw dataset. For the included sample files you don’t need to change anything.

The available settings are: - X dimension: Number of voxels/pixels in the x dimension (this may be the row count) - Y dimension: Number of voxels/pixels in the y dimension (this may be the column count) - Z dimension: Number of voxels/pixels in the z dimension (this may be the slice count) - Bytes to skip: If the dataset contains a header, this should be the size of that header - else it should be 0. - Data format: The format of the voxel data. This will usually be specified somewhere, either in a header or in some documentation for the dataset. You can replace “char” with “int” here (and “uchar” with “uint”). - Endianness: Usually “little endian”

### .ini files

If the folder contains a ".ini" with the same name (dataset.raw => dataset.raw.ini) then this will be used to populate the import settings with some default values. so you don't have to do it every time you import the same dataset.

For an example .ini file, see VisMale.raw.init.

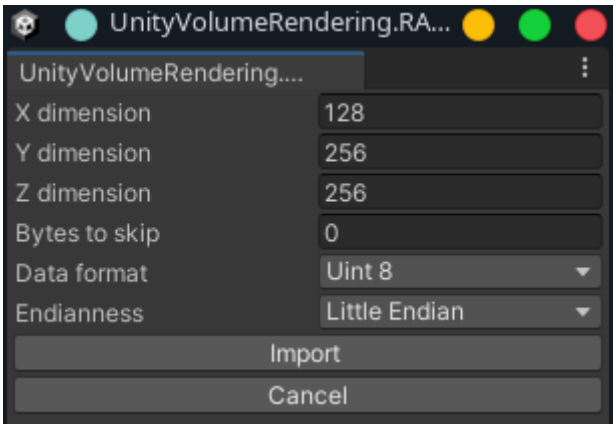


Figure 2:

### DICOM datasets

To import a DICOM dataset, click "Load DICOM" and select the folder where the DICOM dataset is loaded.

If the folder contains multiple series, these will be loaded as separate objects.

**Note:** If you are or Windows or Linux it is recommended to enable the SimpleITK importer, which is a requirement for JPEG2000 compressed DICOM and NRRD.

### NRRD datasets

To import an NRRD dataset, click "Load NRRD dataset" and select the .nrrd file to import.

**Note:** To import NRRD datasets you need to enable the SimpleITK importer.

### NIfTI datasets

To import a NIfTI dataset, click "Load NIFTI dataset" and select the .nii/.nii.gz file to import.

**Note:** If you are or Windows or Linux it is recommended to enable the SimpleITK importer, which should work better overall.

### VASP/PARCHG datasets

To import a VASP dataset, click “Load PARCGH dataset” and select the file to import.

### Image sequence datasets

Some datasets will be stored as a series of image files (similar to DICOM, but in .jpg/.png/.tiff format), where each image represents a slice. These are referred to as “image sequence” datasets.

To import an image sequence, click “Load image sequence” and select the folder containing the slices.

## Importing datasets by drag and drop into the assets folder

To import a RAW/NRRD/NIFTI/PARCHG dataset (not DICOM or other image sequences), you can simply drag and drop the dataset file into the “Assets” directory.

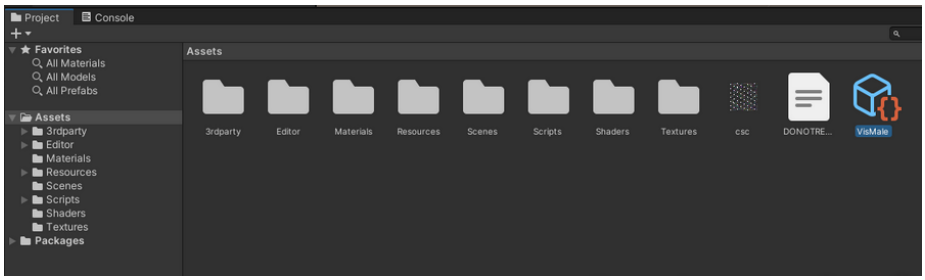


Figure 3:

This will create a new Asset file for the dataset.

For RAW datasets you can change the import settings by clicking on the asset, changing the import settings in the inspector and clicking “Apply”.

To use these datasets, you can either: - Drag and drop the dataset asset into the scene view or scene hierarchy view in the editor (Unity 2021.2 or newer) - Spawn from code: Referencing the `VolumeDataset` asset somewhere in code, or through the Resources folder, and spawn it using the VolumeObjectFactory.

## Importing datasets by right clicking in the Assets folder

You can also import datasets by right clicking in “Assets” directory and choosing “Volume Rendering” -> “Import dataset”

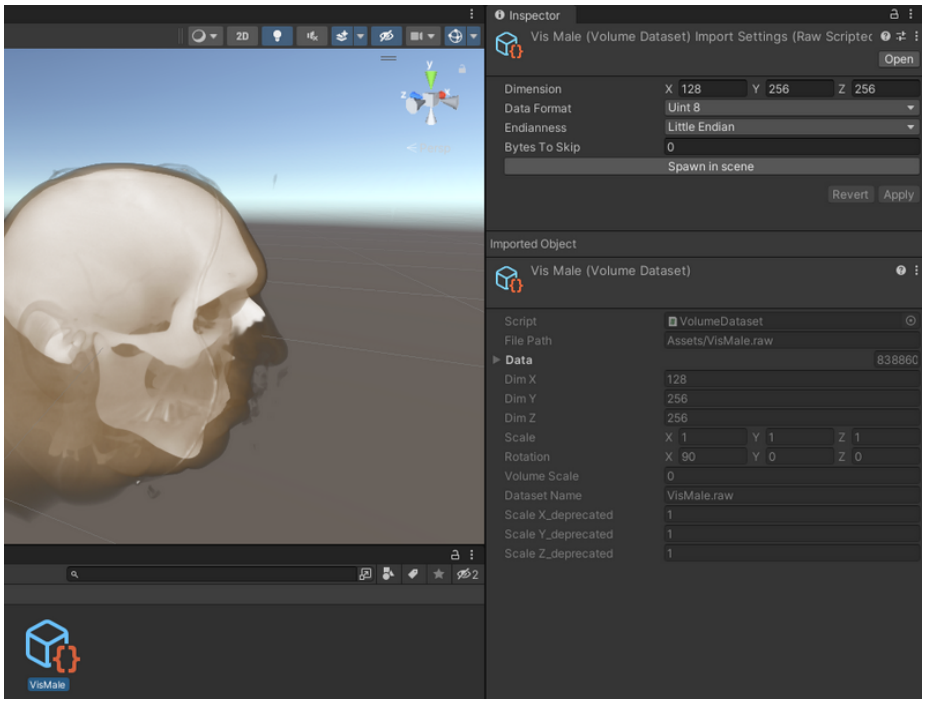
Inspector
Vis Male (Volume Dataset) Import Settings (Raw Scripte
Open
Dimension
X 128
Y 256
Z 256
Data Format
Uint 8
Endianness
Little Endian
Bytes To Skip
0
Spawn in scene
Revert
Apply
Imported Object
Vis Male (Volume Dataset)
Script
VolumeDataset
File Path
Assets/VisMale.raw
Data
Dim X
128
Dim Y
256
Dim Z
256
Scale
Rotation
Volume Scale
0
Dataset Name
VisMale.raw
Scale X_deprecated
1
Scale Y_deprecated
1
Scale Z_deprecated
1
VisMale

Figure 4:

All datasets (also DICOM) can be imported this way.

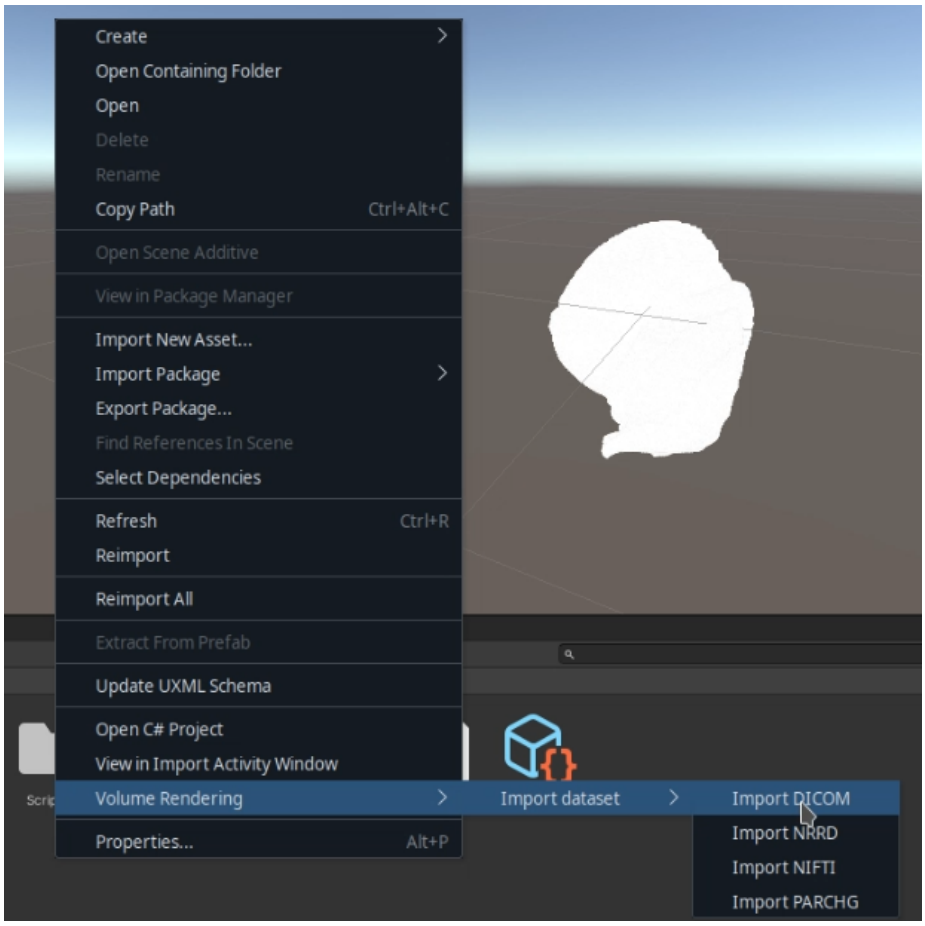


Figure 5:

Using the imported dataset assets works the same as for datasets imported by drag and drop

## Coordinate system and real size

Coordinate systems are handled for DICOM and NRRD, and we convert to Unity's coordinate system using metre units.

By default import datasets will have their size normalised. If you wish to keep the original size (so multiple datasets will fit well together), you can select the

outer GameObject (the ones that has a `VolumeRenderedObject` component), and set its scale to (1,1,1).

## Manipulating spawned datasets

Datasets will be spawned as GameObjects, with a `VolumeRenderedObject` component attached to them.

You can move, rotate and scale these objects like any other GameObject in the Unity Editor.

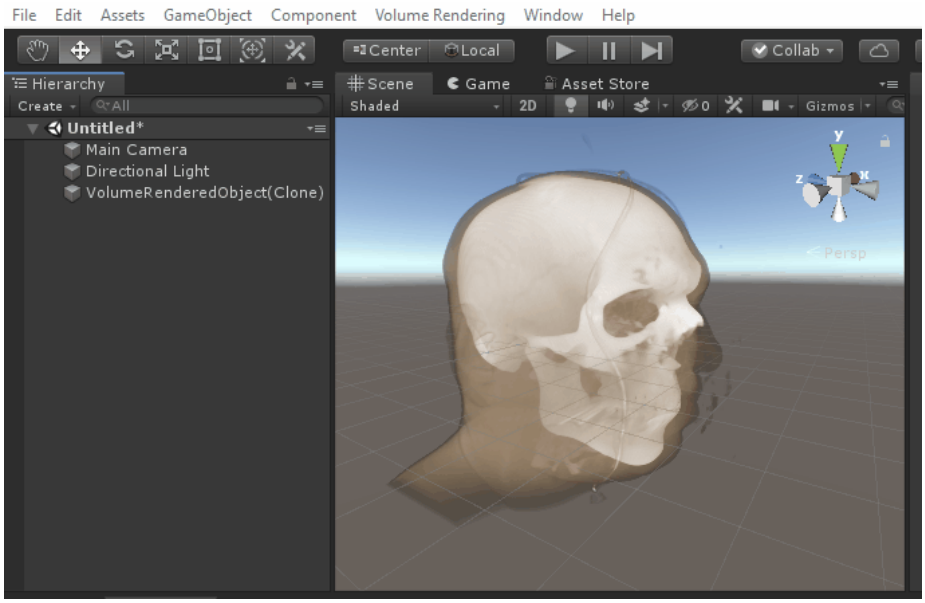


Figure 6:

## Saving spawned datasets

Spawned datasets can be saved as a part of the scene (simply save the scene).

However, if the datasets are large (or many) this will cause the scene asset to become very large, and saving/loading will be slow (or crash!). To prevent this, you may consider importing your datasets as an Asset (see import documentation) and referencing the already imported dataset through some script and spawning it through the VolumeObjectFactory.

# Changing the appearance settings

To change the appearance settings and other volume rendering related settings, see the appearance settings documentation.

# Volume rendering appearance settings

**Table of contents:**



To modify the appearance settings of a volume rendered dataset, simply select the object in the scene / scene hierarchy - and you will find the appearance settings under “Volume Rendered Object” in the inspector.

## Render mode

There are 3 render modes: - Direct Volume Rendering (raymarching, using transfer functions) - Maximum Intensity Projection (shows the maximum density) - Isosurface Rendering (raymarching, stops when it hits a surface)

## Lighting

You can enable lighting to get more realistic visualisation.

This comes at a cost, and performance may suffer (both memory and rendering speed).

To apply lighting to the volume rendering, we calculate the gradient at each voxel and use this to calculate a normal, which we use to apply phong lighting.

## Shadow volumes

Improve the rendered image by using shadow volumes. This is expensive, but there are ways to still get good performance with this enabled. See shadow volume documentation

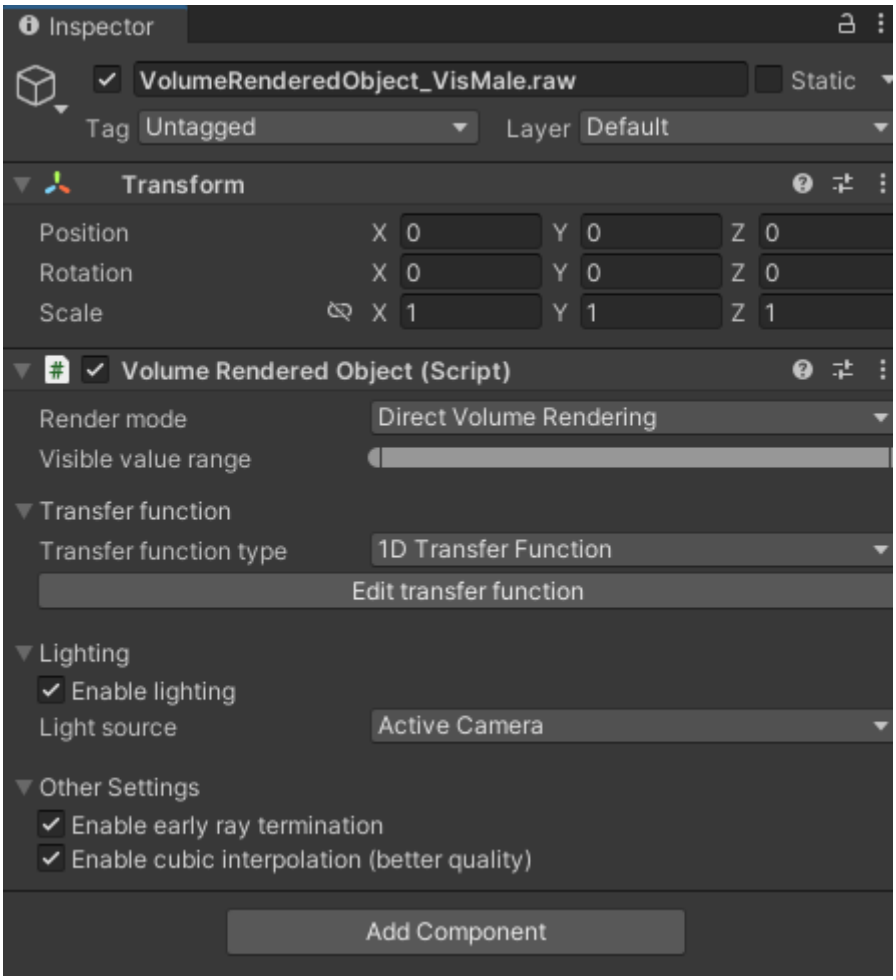


Figure 7:

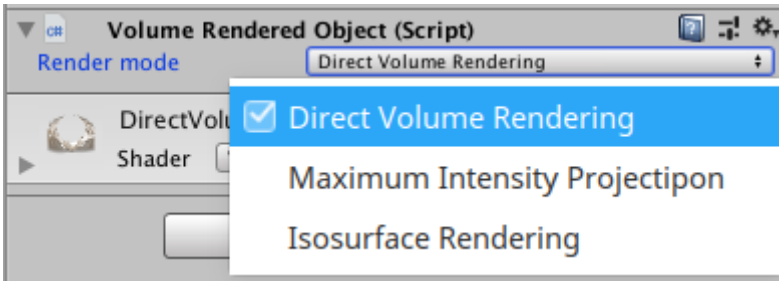


Figure 8:

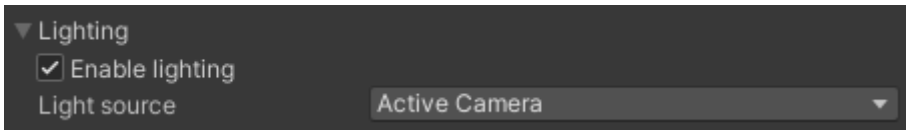


Figure 9:

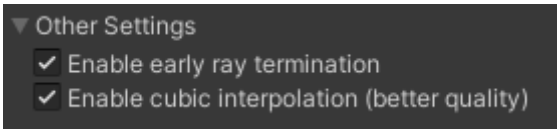


Figure 10:

## Cubic interpolation

To reduce so-called “staircase artifacts”, you can enable cubic interpolation.

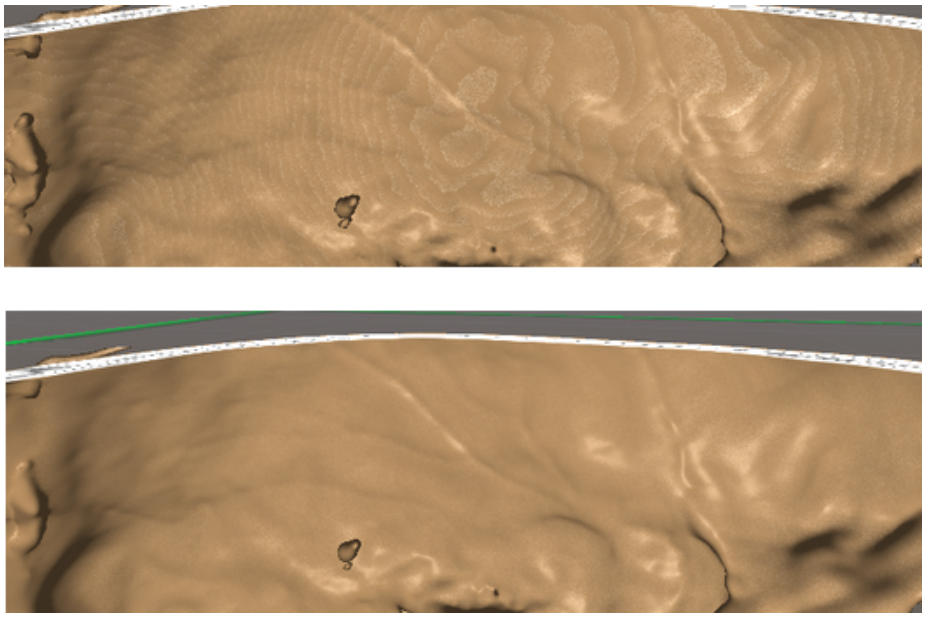

Figure 11:

This can be quite expensive in terms of performance - especially when lighting is enabled!

## Early ray termination

To improve performance, the raymarching shader may optionally exit early when enough samples have been accumulated. This is a very simple optimisation that improves performance by avoiding enumeration of invisible voxels.

You usually want to leave this setting enabled, since it improves performance considerably - usually without any noticable visual changes.

## Shadow volumes

To get more realistic rendering, you can optionally enable shadow volumes.

This can be enabled from the VolumeRenderedObject inspector in the editor, or by adding a `ShadowVolumeManager` component:

### How it works

A shadow volume 3D texture is generated for the whole dataset. A compute shader is repsonsible for updating the shadow volume, by casting rays from the light source through the dataset, and storing information about shadows. Since this can take a long time, and async compute is not available on all platforms, the shadow volume manager divides the shadow volume into smaller chunks, and updates one chunk every frame.

### Performance

Because of the extra work of computing the shadow volume (compute shader) and the extra texture lookups during volume rendreing, this can have a bad impact on performance.

There are luckily some ways to work around this, at least on desktop applications (Windows, Linux, etc.). - If you're using HDRP: Enable DLSS and reduce the render scale. - If you're using URP: Enable FidelityFX Super Resolution and reduce the render scale.

This works by rendering to a smaller render target (which is usually the bottleneck during volume rendering) and then doing "magic" upscaling on the rendered image.

# Transfer functions

You can use transfer functions to apply different colours and opacities to different parts of the dataset, resulting in more interesting visualisations.

Ttransfer functions will map density (and optionally gradient magnitude) to a selected colour/opacity, enabling you to highlight parts of the dataset (skin, bones, etc.).

You can open the transfer function editors from the “Volume Rendering” menu bar option in the editor:

## 1. 1-Dimensional transfer functions

This is the simplest type of transfer function to work with. It maps density values to a selected colour and opacity (transparency).

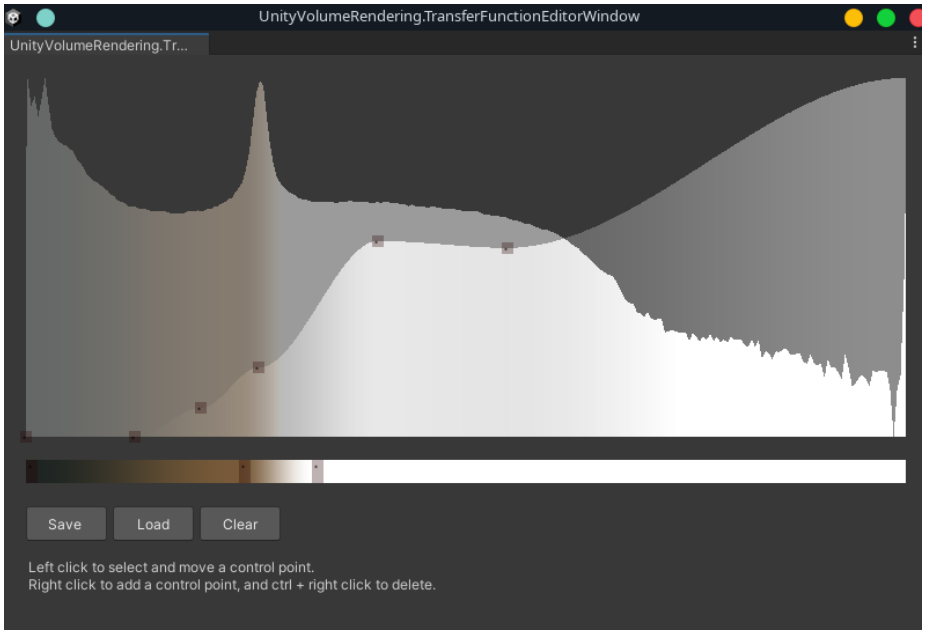


Figure 12:

The axes: - X-axis: Represents density values. - Y-axis: Represents alpha values (opacity).

How to use: - Move the grey alpha knots to create a curve for opacity by density. - Right-click to add new alpha knots. - The bottom gradient-coloured panel

maps colour to density. Here you can select which colour a part of the dataset should have, based on the density values of the voxels. - Right-click to add new knots and click on an existing colour knot to modify its colour.

### 2. 2-Dimensional transfer functions

**Note: The 2D TF editor is still work-in-progress.** I plan to improve it further. Suggestions and feedback is very welcome! (create an issue or start a discussion thread).

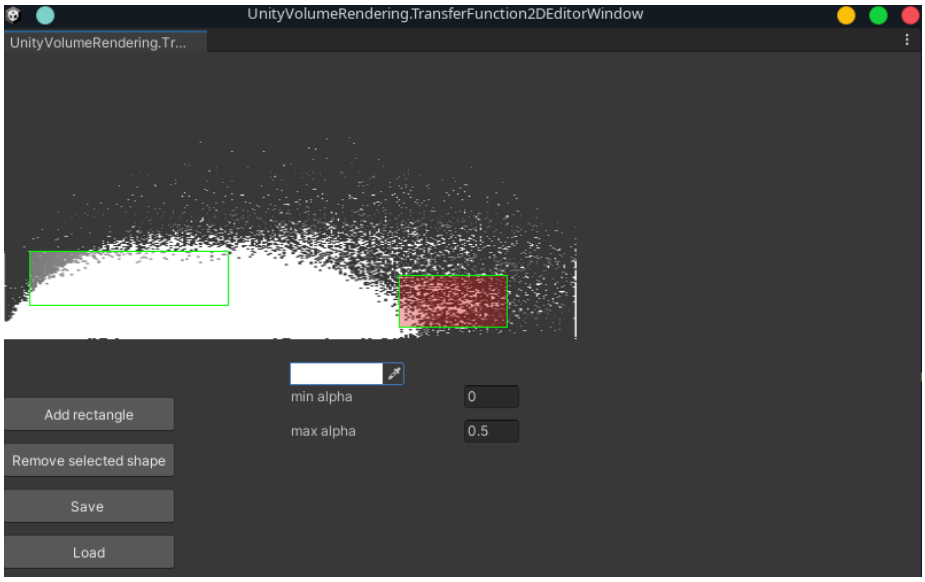


Figure 13:

The axes: - X-axis: Represents density - Y-axis: Represents gradient magnitude.

How to use: - Click “add rectangle” to add a new rectangle-shape. - Click on the rectangle and drag to move it. - Drag the edges of the rectangle to change its shape. - Click on the colour picker to change the colour of the selected rectangle shape. - Change the “min alpha” and “max alpha” sliders to change the minimum and maximum opacity of the selected shape. - The “max alpha” defines the opacity value at the centre of the shape. - The “min alpha” defines the opacity value at the edges of the shape.

## Cross section tools

**Table of contents:**

- Cross section tools
  - Cross section plane
  - Box cutout
  - Sphere cutout

You can spawn cross section tools from the Volume Rendering menu option.

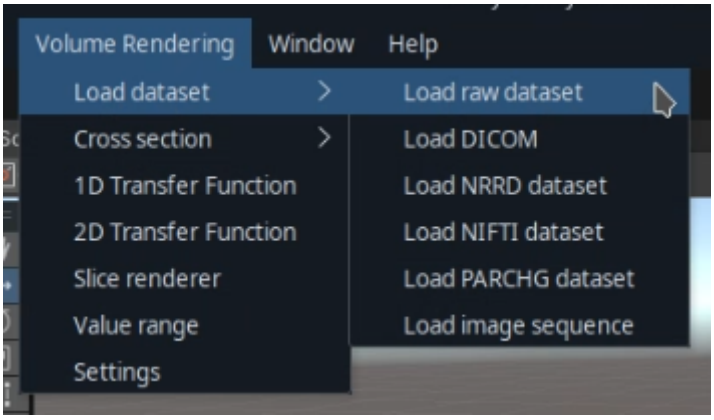


Figure 14:

There are three types of cross section tools:

## Cross section plane

The cross section plane is a simple plane that can be moved around and rotate, to clip the volume at an arbitrary position and angle.

If you need axis aligned clipping planes, I would recommend to implement your own clipping plane tool that implements the `CrossSectionObject` interface, and add it to the `CrossSectionManager`.

## Box cutout

The box cutout tool spawns a box that you can move around and roate to do a box cutout on a dataset.

The tool has two modes: - Exclusive: Cuts out everything that overlaps the box. - Inclusive: Cuts out everything that is outside the box.

You can change these modes by selecting the “CutoutBox” GameObject in the hierarchy and changing the settings in the inspector:

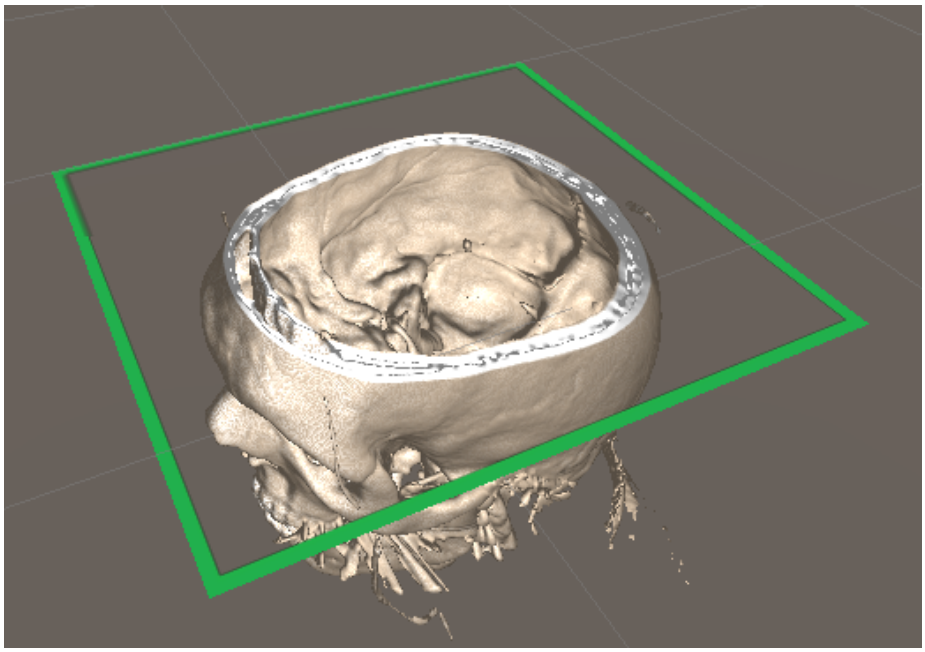

Figure 15:

### Sphere cutout

The sphere cutout tool spawns a sphere that you can move around and roate to do a sphere cutout on a dataset.

The tool works the same as the box cutout, and also has two modes: inclusive and exclusive.

## Slice renderer

The slice renderer can be used to create a axis aligned (or free transform) slice view of a dataset.

### How to open the slice renderer window

You can open the slice renderer window from the “Volume Rendering” menu bar option:

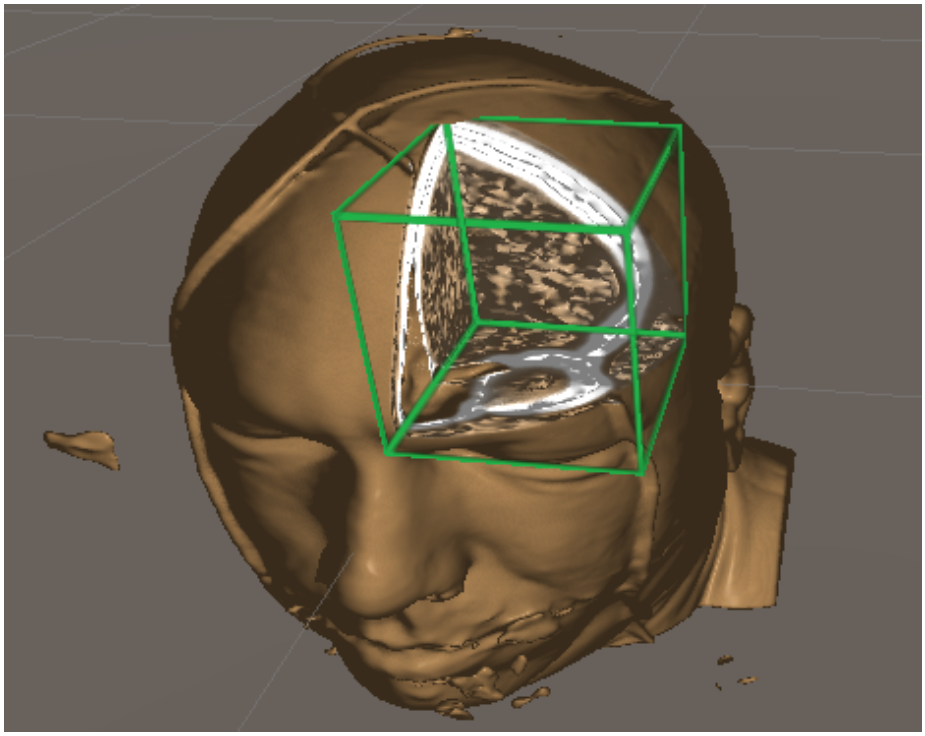

Figure 16:


Figure 17:

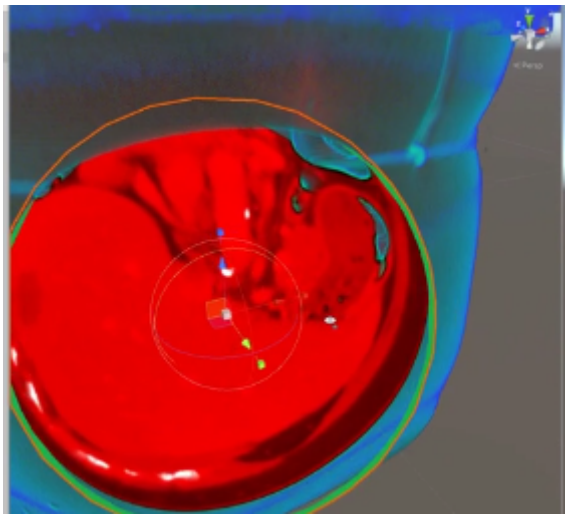

Figure 18:

### Creating slicing planes

To create a new slice renderer, click one of the “Create plane” buttons.

The buttons to the left can be used to navigate between different slices.

### Rotating the slice view

At the top of the window there are some more buttons.

The two rightmost buttons can be used to rotate the slice view 90 degrees.

### Moving and measuring the slices

The three buttons to the top left will change the current interaction mode: - **Move slice**: Left click + drag mouse up/down => Slice moves forwards/backwards - **Inspect values**: Left click somewhere in the view, and the value will be printed in the bottom left corner. - **Measure distances**: Left click and drag mouse to measure a linear distance. The value will be printed at the bottom left corner.

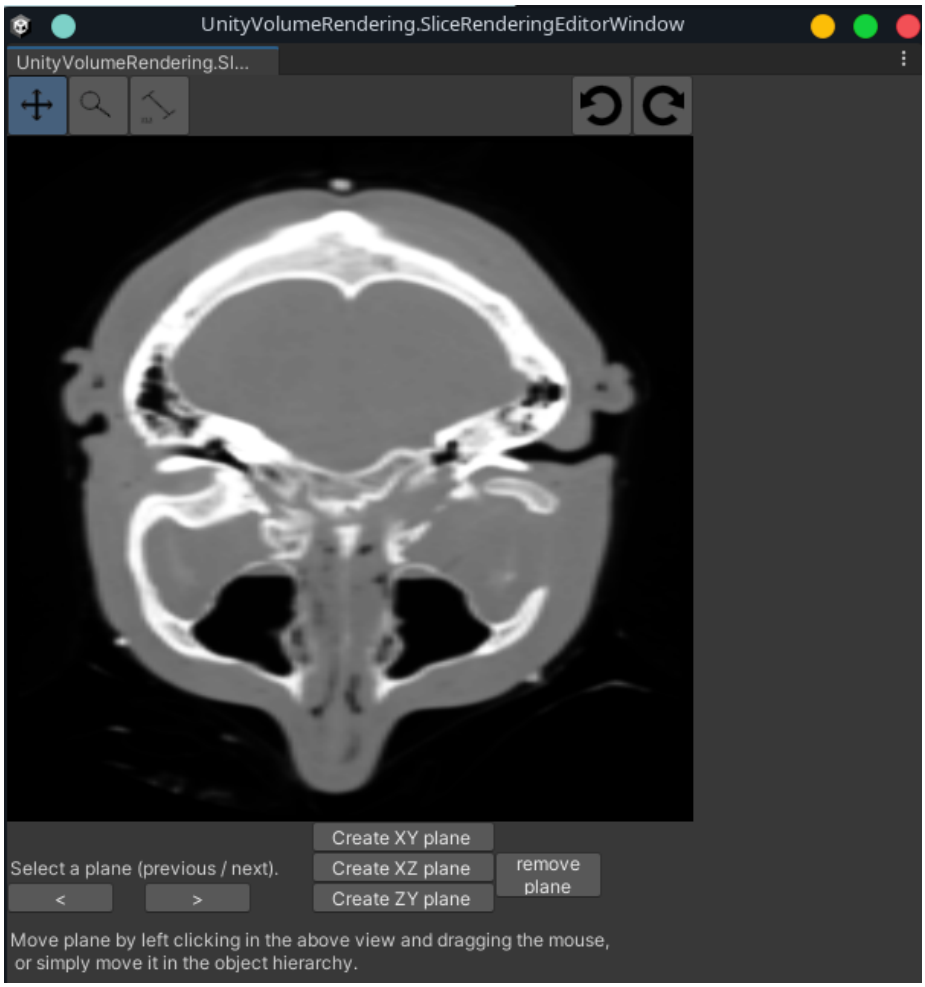


Figure 19:

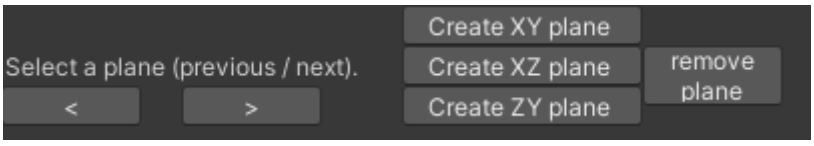


Figure 20:


UnityVolumeRendering.SliceRenderingEditorWindow
UnityVolumeRendering.Sl...
Create XY plane
Create XZ plane
Create ZY plane
remove plane
Select a plane (previous / next).
<
>
Move plane by left clicking in the above view and dragging the mouse,
or simply move it in the object hierarchy.

Figure 21:

# Settings window

This window exposes some dataset and import-related settings, for using the plugin in the Editor.

## How to open the settings window

You can open the settings window from the "Volume Rendering" menu bar option:

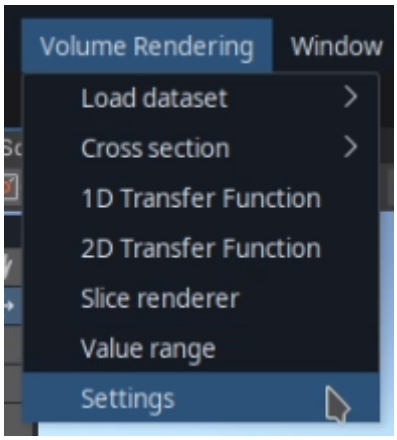


Figure 22:

## Available settings

### Show downscale prompt

When this setting is enabled, you will see a promt asking you if you wish to downscale the dataset every time you import a dataset in the editor.

Imported datasets are automatically downscaled if they are too large to fit into a 3D texture. However, in some cases you may wish to downscale it even if this is not a problem (to save memory, etc.).

### Enable SimpleITK

If you're on Windows/Linux/Mac, you can optionally enable the SimpleITK-based importer.

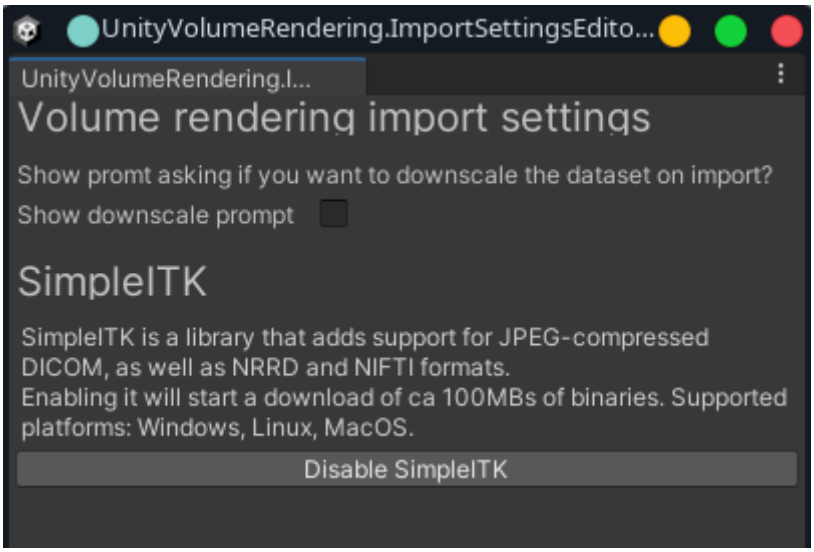


Figure 23:

This has a couple of advantages: - Better DICOM support (faster import, and fewer issues) - JPEG2000 compressed DICOM support - NRRD support - Better NIFTI support - Support for image sequence datasets in TIFF format

It is recommeneded to enable this if you don't only use RAW datasets.

This is a native plugin, so to support other platforms you would need to build it yourself.

## Importing datasets form code

**Table of contents:**



There are 3 types of importers: - Raw importer - Used for importing raw binary datasets. - These datasets can optionally have a header, followed by raw 3D data in various formats (int, uint, etc.). - Image file importer - Used for importing a single file dataset. - Supported formats: VASP, NRRD; NIFTI - Image sequence importer - Used for importing sequences datasets, where each slice maybe be stored in a separate file (multiple files per dataset).

## Raw importer

The *RawDatasetImporter* imports raw datasets, where the data is stored sequentially. Some raw datasets contain a header where you can read information about how the data is stored (content format, dimension, etc.), while some datasets expect you to know the layout and format.

To import a RAW dataset, do the following:

```
// Create the importer
RawDatasetImporter importer = new RawDatasetImporter(filePath, initData.dimX, initData.dimY,
// Import the dataset
VolumeDataset dataset = importer.Import();
// Spawn the object
VolumeObjectFactory.CreateObject(dataset);
```

The *RawDatasetImporter* constructor takes the following parameters: - filePath: File path to the dataset. - dimX, dimY, dimZ: The dimension of the dataset. - contentFormat: The format of the content. Possible values: Int8, Uint8, Int16, Uint16, Int32, Uint32. - endianness: The byte endianness of the dataset. - skipBytes: Number of bytes to skip before reading the content. This is used in cases where the dataset has a header. Some raw datasets formats store information about the dimension, format and endianness in a header. To import these datasets you can read the header yourself and pass this info to the *RawDatasetImporter* constructor. The skipBytes parameter should then be equal to the header size.

All this info can be added to a ".ini"-file, which the importer will use (if it finds any). See the sample files (in the "DataFiles" folder for an example).

## Image file importer

To import single-file datasets, such as VASP/PARCHG, NRRD and NIFTI, you can use one of the image file importers. You can manually create an instance of your desired importer, or you can simply use the `ImporterFactory` class, which will select one for your desired file format.

Example:

```
IImageFileImporter importer = ImporterFactory.CreateImageFileImporter(ImageFileFormat.NRRD);
VolumeDataset dataset = importer.Import(file);
VolumeRenderedObject obj = VolumeObjectFactory.CreateObject(dataset);
```

Possible parameters to *ImporterFactory.CreateImageFileImporter*: - ImageFileFormat.NRRD (requires SimpleITK) - ImageFileFormat.NIFTI (requires SimpleITK) - ImageFileFormat.VASP

The available importer implementations are: - *ParDatasetImporter*: For VASP/PARCHG. - SimpleITKImageFileImporter: For NRRD and NIFTI.

Works on Windows and Linux (and hopefully MacOS too).

For more information about NRRD support, see the page about SimpleITK.

## Image sequence importer

To import an image sequence dataset, such as DICOM, you can manually create an instance of one of the image sequence importers or simply use the `ImporterFactory` class, which will select one for you.

Example:

```
// Get all files in DICOM directory
List<string> filePaths = Directory.GetFiles(dir).ToList();
// Create importer
IImageSequenceImporter importer = ImporterFactory.CreateImageSequenceImporter(ImageSequenceF
// Load list of DICOM series (normally just one series)
IEnumerable<IImageSequenceSeries> seriesList = importer.LoadSeries(filePaths);
// There will usually just be one series
foreach(IImageSequenceSeries series in seriesList)
{
    // Import single DICOm series
    VolumeDataset dataset = importer.ImportSeries(series);
    VolumeObjectFactory.CreateObject(dataset);
}
```

These importers can import one or several *series*. In most cases there will only be one series. However, in DICOM each DICOM slice can be associated with a "series". This allows you to store several datasets in the same folder.

Supported formats: - ImageSequenceFormat.DICOM - ImageSequenceFormat.ImageSequence

The available importer implementations are: - SimpleITKImageSequenceImporter: For DICOM (see SimpleITK.md for more info.) - DICOMImporter: For DICOM. Uses OpenDICOM library, and works on all platforms. This is the default when SimpleITK is disabled. - ImageSequenceImporter: For image sequences (directory containing multiple image files, typically JPEG or PNG)

### Notes about DICOM support

The SimpleITK-based importer is the recommended way to import DICOM datasets, as it supports JPEG compression. See the SimpleITK documentation for information about how to enable it. Once enabled, *ImporterFactory.CreateImageSequenceImporter* will automatically return an importer of type `SimpleITKImageSequenceImporter`.

## Async import

Most of the importers also support asynchronous import. This is very useful for VR/AR applications where you defnitely don't want the import to freeze the whole application for too long.

To do async import, create and run an async Task that calls the `Async` version for the importer factory's import methods. Below is an example:

```
private static async Task DicomImportDirectoryAsync(IEnumerable<string> files)
{

    using (ProgressHandler progressHandler = new ProgressHandler(new EditorProgressView()))
    {
        progressHandler.StartStage(0.2f, "Loading DICOM series");

        IImageSequenceImporter importer = ImporterFactory.CreateImageSequenceImporter(ImageS
        IEnumerable<IImageSequenceSeries> seriesList = await importer.LoadSeriesAsync(files,

        progressHandler.EndStage();
        progressHandler.StartStage(0.8f);

        int seriesIndex = 0, numSeries = seriesList.Count();
        foreach (IImageSequenceSeries series in seriesList)
        {
            progressHandler.StartStage(1.0f / numSeries, $"Importing series {seriesIndex + 1
            VolumeDataset dataset = await importer.ImportSeriesAsync(series, new ImageSequer
            progressHandler.EndStage();
        }

        progressHandler.EndStage();
    }
}
```

You can optionally pass in a progress handler, which is used to track the progress of the async import. The `ProgressView` is used to display the progress, either in the Unity Editor or in your own GUI. In the above example we use the `EditorProgressView`, which will show a progress bar in the editor - but you can also create your own.

## Volume raycasts (finding intersections with the volume)

To find ray intersections with the volume, for example to find out where on the volume the user has clicked, you can use the `VolumeRaycaster` class.

For an example, see the `DistanceMeasureTool` class, which can be used from the SampleScene.

### Example:

```
// Create a ray form where the user clicked
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
// Create a raycaster instance
VolumeRaycaster raycaster = new VolumeRaycaster();
// Raycast the scene, with our ray. The "hit" output variable will contain the result, if an
if (raycaster.RaycastScene(ray, out RaycastHit hit))
{
    // Debug draw a line representing the ray (from eye to hit point). Only visible in the s
    Debug.DrawLine(ray.origin, hit.point, Color.red, 10.0f, true);
}
```

## SimpleITK

SimpleITK is a library that supports a wide range for formats, such as: - DICOM (with JPEG compression) - NRRD - NIFTI

This project optionally uses SimpleITK for the above formats. There is another fallback DICOM importer, but SimpleITK is a requirement for NRRD and for JPEG2000 compressed DICOM datasets.

Since SimpleITK is a native library, that requires you to download some large binaries for each target platform, it has been disabled by default.

To enable SimpleITK, you simply have to do the following: 1. In Unity's top toolbar, click "Volume rendering" and then "Settings", to open the settings menu. 2. In the settings menu, click "Enable SimpleITK"


This will automatically download the SimpleITK binaries, and enable support for SimpleITK in the code. The `ImporterFactory` class will then return the SimpleITK-based importer implementations.

### Supported platforms

Currently the SimpleITK integration supports Windows, Linux and MacOS. To use it on other platforms you could probably try building the official C# wrapper for that platform, or manually download the SimpleITK binaries for that platform and create your own C# wrapper. However, I'll look into distributing binaries for at least Linux (which is what I use as a daily driver).

Note: If you wish to enable SimpleITK, you currently need to create the build on the same platform as your target platform. If you wish to create a Linux build on Windows, you would need to manually download the SimpleITK Linux binaries before you build.