LISCINTEC
Volume Viewer Pro

Documentation

LISCINTEC
Volume Viewer Pro

1. What is Volume Viewer Pro?

Volume Viewer Pro is a shader based package from the Unity Asset Store. It enables the customer to visualize volumetric data (e.g. Texture3D assets) in real time using volume ray casting (aka volumetric ray tracing or volume ray marching). Volume Viewer Pro can render any kind of scalar or RGBA data that can be expressed as a three dimensional array, including MRI and CT scans as well as microscopy slices. It can be set up in less than a minute and is easy to use.

Prominent features:
- supports rendering multiple volumes.
- supports rendering on multiple cameras.
- opaque objects obscure volumes.
- 1D or 2D transfer function for volumes.
- overlays to highlight regions of interest.
- 1D transfer function (TF) for overlays to highlight distinct regions differently.
- Blinn Phong shading for light interactions.
- 2D TF for custom light models.
- compatible with SteamVR and OpenVR.*
- reduction of wood grain artifacts by using a ray offset texture.
- no clipping errors when the camera enters a volume.
- various settings to alter image quality and performance.
- increased performance compared to the basic Volume Viewer package.
- various real time render settings:
    - number of samples along the ray.
    - contrast / brightness / opacity.
    - rescaling values.
    - cut primitive shapes out of volumes (by excluding voxels that intersect with objects that have a special material on them).

Loads data from various file types: **
- Texture3D asset files.
- PNG files.
- JPG files.
- NIfTI files.***
- DICOM files (limited to Transfer Syntax 1.2.840.10008.1.2 and 1.2.840.10008.1.2.1).***

LISCINTEC
Volume Viewer Pro

Not suited for mobile!
Not suited for large data sets!

While Volume Viewer Pro enables customers to load and render 3D medical scans, we want to emphasize that this package is not approved for any medical purpose.

* Volume ray casting is resource intensive, especially in VR. We do not guarantee a minimum frame rate.
** No file support other than Texture3D assets in WSA (NETFX_CORE).
*** Voxel data is loaded as is, independent of header information. This can result in mirror-inverted images. A negative transform scale can correct for this but needs to be applied manually.

2. What does Volume Viewer Pro do?

Volume Viewer Pro utilizes volume ray casting to create a projection of volumetric data. Subsequently, the projection is blended on top of the final camera image.
When creating the projection, the depth of opaque objects in the scene is taken into account to create the illusion that they can obstruct the view of volume objects. If multiple volume objects are to be rendered by the same camera, multiple such projections are made - each one made of only one volume object - and blended on top of the final camera image in the order of their depth to the camera (from furthest to closest) to abide by their relative positions in scene space. If voxels of volume objects would intersect with the mesh of an object that has the CullingMaterial on it (from now on referred to as culling objects), they are excluded from the projection. This effect can also be inverted, so that only voxels that intersect with a culling object are projected onto the camera's image.

3. How to set up Volume Viewer Pro?

Volume Viewer Pro can be set up in a few simple steps:

1. Import the Volume Viewer Pro package.
2. Add a User Layer called 'VolumeViewer' (without the quotes).
   You can add a User Layer by clicking on 'Layers' (in the upper right corner of the Unity Editor) and choosing 'Edit Layers...'.
3. Make sure that Volume Viewer Pro's shaders are going to be included in a build by going to Edit → Project Settings → Graphics. Increase the size of the AlwaysIncludedShaders array by 8 and add all 8 shaders in Assets/VolumeViewerPro/shaders/ to the array.

This is all you need to do to get most of the example scenes up and running.

To use Volume Viewer Pro in a VR environment, follow the steps in 'How to set up Volume Viewer Pro with VR?'.

To use Volume Viewer Pro in a new scene, follow the steps in 'How to use Volume Viewer Pro in a new scene?'.

4. How to set up Volume Viewer Pro with VR?

You can use Volume Viewer Pro in a VR environment by following these simple steps:

1. If you haven't already, set up Volume Viewer Pro by following the steps in 'How to set up Volume Viewer Pro?'
2. Properly connect your VR device (HMD).
3. Start the necessary software to use your VR device (e.g. SteamVR).
4. Depending on the Unity version you are using:
    - Unity 5.3.6:
        1. Download and import the SteamVR Plugin from the Unity Asset Store.
        2. Drop the [CameraRig] prefab (in Assets/SteamVR/Prefabs/) into your scene.
        3. In the Hierarchy view, select the 'Camera(eye)' object (found at '[CameraRig]' → 'Camera(head)' → 'Camera(eye)').
        4. In the inspector, add the VolumeRenderer component to the 'Camera(eye)' object.
        5. In the VolumeRenderer component, set the size of the 'VolumeObjects' array to 1 and drag a cube with a VolumeComponent component into the slot of 'Element 0'. This enables the camera to render that particular volume object.
    - Unity 5.4 or above:
        1. Go to 'Edit' → 'Project Settings' → 'Player'.
        2. In 'Other Settings' enable 'Virtual Reality Supported'
        3. In the 'Virtual Reality SKDs' menu, add the SDK that best supports your VR device (e.g. OpenVR for HTC Vive) and move it to the top of the list.
        4. Depending on your preferred camera:
            - SteamVR Plugin:
                1. Follow the same steps as for Unity 5.3.6, outlined above.
            - Regular Unity Camera (e.g. Main Camera):
                1. In the Hierarchy view, select your camera.
                2. In the inspector, add the VolumeRenderer component to your camera.
                3. *Continue on the next page...*

4. In the VolumeRenderer component, set the size of the 'VolumeObjects' array to 1 and drag a cube with a VolumeComponent component into the slot of 'Element 0'. This enables the camera to render that particular volume object.

5. How to use Volume Viewer Pro in a new scene?

You can use Volume Viewer Pro in a new scene by following these simple steps:

1. If you haven't already, set up Volume Viewer Pro by following the steps in 'How to set up Volume Viewer Pro?'
2. Create a new standard cube. You can create a cube by clicking on GameObject → 3D Object → Cube.
3. If you want to visualize volumetric data from a Texture3D asset, add the script VolumeComponent (in Assets/VolumeViewerPro/scripts/volume/) to the cube and drag the Texture3D asset into the 'Data' field in the inspector. Check the 'ActivateOnStartup' checkbox and continue with 6.
4. If you want to visualize volumetric data from a PNG, JPEG, NifTI or DICOM file, add the script VolumeFileLoader (in Assets/VolumeViewerPro/scripts/files/) to the cube and set the 'DataPath' field in the inspector to the path of the file. Continue with 6.
5. If you want to visualize volumetric data previously saved to a VolumeComponentAsset add the script VolumeFileLoader (in Assets/VolumeViewerPro/scripts/files/) to the cube and drag the VolumeComponentAsset into the 'LoadAssetInstead' field in the inspector.
6. Add the script VolumeRenderer (in Assets/VolumeViewerPro/scripts/camera/) to the camera. In the inspector, set the size of the 'VolumeObjects' array to 1 and drag the cube into the slot of 'Element 0'. This enables the camera to render that particular volume object.

6. Example scenes.

The following example scenes are included in Volume Viewer Pro:

1. MinimalTexture3D
   A simple example of how to render a Texture3D asset. It includes a cube with a VolumeComponent which is partially obstructed by two standard cubes. A third cube demonstrates how to use the CullingMaterial, to exclude voxels that intersect with it in real time.

2. Procedural
   An example of how to procedurally generate volumetric data and how to animate features of a VolumeComponent.

3. MinimalNIfTI
   Simple example of how to render scalar volumetric data from a NIfTI file using the VolumeFileLoader.

4. MRI_UI
   Example of an MRI Viewer. A cube with a VolumeFileLoader will load the data of an MRI scan from a NIfTI file. UI elements can be used to manipulate how the volume object is rendered in real time.

5. Lighting
   Simple example of how to render scalar volumetric data from a NIfTI file using the Blinn Phong shading model.

6. LightingTF
   Simple example of how to render scalar volumetric data from a NIfTI file using a custom lighting transfer function.

7. Overlay
   Example of how to create and animate a transfer function for an overlay.

8. MultiObjMultiCam
   Example of how to render two different volume objects onto two different cameras. The two volume objects rotate around a moving cube to demonstrate correct depth rendering from every angle.

9. MinimalVR
   Simple example of how to render scalar volumetric data from a NIfTI file in VR.

7. Important scripts and their members.

There are 3 main scripts in Volume Viewer Pro:

VolumeFileLoader:

Loads volumetric data from files. If a data set consists of multiple files, all those files should be moved in one folder that includes no other files. To set a correct path, click on 'Choose' and select one of those files. It does not matter which file was selected. The order in which they will be read is determined by their name or their header information. Thus a consistent naming scheme is recommended.

Important members:

loadOnStartup:	Start loading in Start().Otherwise loadFiles() can be called later by another script.
dataPath:	Path to the file that contains the volumetric data.
forceDataFormat:	Force a particular data format.
forceDataPlanarConfiguration:	Force a particular planar configuration in case the data is RGBA. If the correct planar configuration isn't inferred from the header, this can be used to load color channels in the desired order. You should use this if you see duplicates (3 or 6) of your data.
overlayPath:	Path to the file that contains the volumetric overlay.
forceOverlayFormat:	Force a particular overlay format.
forceOverlayPlanarConfiguration:	Force a particular planar configuration in case the overlay is RGBA. If the correct planar configuration isn't inferred from the header, this can be used to load color channels in the desired order. You should use this if you see duplicates (3 or 6) of your overlay.
loadAssetInstead:	A reference to a VolumeComponentAsset (cache) that should be loaded instead of dataPath or overlayPath.
loadingProgressChanged:	This event will send the current progress (between 0 and 1) of loading the data (and if present, the overlay) to subscribers.

VolumeComponent:

Includes a collection of members that determine the appearance of the volume projection. Not unlike a Material. Most of them have Events to notify subscribers of any changes.

Important members:

resolution:	Image resolution when rendering (divisor for screen width and height). Increasing this value will decrease quality but increase performance.
maxSamples:	Maximum number of values that are sampled along the ray. Increasing this value will increase quality but decrease performance.
rayOffset:	Should be a texture with random noise in channel R. Used to reduce wood grain artifacts.
data:	Volumetric data.

dataType:	Reports if the voxels of the volumetric data are scalar or RGBA values. Not visible in the inspector.
voxelDimensions:	Grid spacing (width, height and depth of a single voxel).
dataChannelWeight:	A weight for the channels of the volumetric data. Only visible when dataType is RGBAData.
valueRangeMin:	Rescale values according to the formula: (max(voxel.rgb) - valueRangeMin) / (valueRangeMax - valueRangeMin).
valueRangeMax:	Rescale values according to the formula: (max(voxel.rgb) - valueRangeMin) / (valueRangeMax - valueRangeMin).
cutValueRangeMin:	Before rescaling, set values below cutValueRangeMin to 0.
cutValueRangeMax:	Before rescaling, set values above cutValueRangeMax to 0.
tfData:	Transfer function that should be used on the data.
tfDataBlendMode:	How should the transfer function interact with the data?
tf2D:	Is the second dimension of the transfer function (gradient magnitude) needed? Setting tf2D to false will assume a gradient magnitude of 0 for each voxel. Setting tf2D to true will require calculation of gradient magnitudes for each voxel within the shader. This will decrease performance.
gradientRangeMin:	Rescale gradient magnitude values according to the formula: (magnitude - gradientRangeMin) / (gradientRangeMax - gradientRangeMin).
gradientRangeMax:	Rescale gradient magnitude values according to the formula: (magnitude - gradientRangeMin) / (gradientRangeMax - gradientRangeMin).
cutGradientRangeMin:	Before rescaling, set magnitudes below cutGradientRangeMin to 0.
cutGradientRangeMax:	Before rescaling, set magnitudes above cutGradientRangeMax to 0.
overlay:	Volumetric overlay (e.g. fMRI ROI). Used to distinguish certain voxels from others by giving them the same value in the volumetric overlay.
overlayType:	Reports if the voxels of the volumetric overlay are scalar or RGBA values. Not visible in the inspector.
overlayChannelWeight:	A weight for the channels of the volumetric overlay. Only visible when overlayType is RGBAData.
overlayBlendMode:	How should the overlay interact with the data?
overlayVoidsCulling:	Should a visible voxel of the overlay be prevented from being culled?
tfOverlay:	Transfer function that should be used on the overlay.
tfOverlayBlendMode:	How should the transfer function interact with the overlay?
contrast:	Change the VolumeComponent's contrast according to the formulas: contrastValue = 1.1 * (contrast + 1.0) / (1.0 * (1.1 - contrast)) rgb = contrastValue * (rgb - 0.5) + 0.5 + brightness
brightness:	Change the VolumeComponent's brightness according to the formula rgb = contrastValue * (rgb - 0.5) + 0.5 + brightness.
opacity:	Change the VolumeComponent's opacity.
enableLight:	Enable interaction with a directional light.
surfaceThr:	Threshold value to determine the iso surface that shall interact with light.
surfaceAlpha:	Alpha value of the iso surface and its interaction with light.
ambientColor:	Ambient color.
diffuseColor:	Diffuse color.
specularColor:	Specular color.
shininess:	Shininess of specular part.

maxShadedSamples: If surfaceAlpha is below 1, more than 1 iso surface could be deteted along the ray. How many iso surfaces should be shaded along the ray? Increasing this value will decrease performance.

surfaceGradientFetches: Determines the smoothness of the gradient used for light interaction. Increasing this value will increase visual quality but decrease performance.

tfLight: Instead of using the Blinn-Phong lighting model, use a custom-lighting transfer function. n∗l vs n∗h.

hideZeros: Set the alpha of values that are 0.0 to 0.0.

invertCulling: Invert the culling effect of objects with a CullingMaterial on them.

VolumeRenderer:

Renders the volume projection and blends it on top of the final camera image.

Important members:

enableCulling: Should the camera consider or ignore culling objects? If you don't need culling, setting enableCulling to false will increase performance.

enableDepthTest: Should the camera consider or ignore the depth of opaque objects in the scene? If you don't need scene integration, setting enableDepthTest to false will increase performance.

leap: How many samples should be skipped if the intensity is very small? Increasing this value will decrease quality but increase performance.

volumeObjects: List of VolumeComponents to be rendered.

8. Troubleshooting.

The volume object keeps showing 'Loading Volume. Please wait.' and nothing happens.

The volume object will only be rendered in the game-view (during play-mode) or in a build. If you are in a build, make sure you set up Volume Viewer Pro correctly (see 'How to set up Volume Viewer Pro?'). If the texture is red instead of white, the volumetric data could not be loaded. If you used a relative path (as opposed to an absolute path) in a build and executed the build from outside your project folder, a relative path will no longer be valid.

Low frame rates or flickering.

Volume ray casting is a computationally intense method. The more often the ray is sampled, the more computations need to be performed. If you encounter low frame rates or flickering, it is likely that the amount of samples you chose on the VolumeComponent, is to high for your hardware to render at the desired frame rate. You can decrease the number of samples by changing the 'Max Samples' variable in the inspector of the VolumeComponent. Decreasing 'Max Samples' will decrease the rendering quality of the volume object but increase performance and vice versa. Another possibility to increase performance is to increase the 'Resolution' variable which, in turn, also decreases visual quality. If scene integration or culling isn't required, you can deactivate it in the inspector of the VolumeRenderer script. This will further increase performance.

There is just a black cube.

If you see only a black cube, try changing the 'CutValueRangeMin' variable or other members of the VolumeComponent. Also, make sure 'HideZeros' is set to true.

There is just a pink cube.

There might be a compatibility issue between Unity versions. Please click on each of the shader files in the Project view and report any errors or warnings you might see to info@liscintec.com

The overlay doesn't look right.

Make sure that the FilterMode is set to Point for both, the overlay and the overlay's transfer function. Otherwise, interpolation between different IDs in the overlay or between adjacent colors in the overlay's transfer function will create visual artifacts. WrapMode should be set to Clamp.

The cube vanishes and nothing is rendered in its place.

Make sure you included our shaders in the 'AlwaysIncludedShaders' array as described in 'How to set up Volume Viewer Pro'. If this doesn't help, try to set your RenderingPath to Forward in the Tier settings at Edit → Project Settings → Graphics.

Lighting doesn't look as smooth as in the video!
GradientFetches 32 and 54 don't work!

Calculating smooth gradients for light interactions takes a long time to compile for all the different shader variants (about an hour on our machines). To keep compile time short, by default, we excluded the smooth gradient calculation from the shader. To enable it, uncomment one or both of these lines in Assets/VolumeViewerPro/shaders/

```
#define VOLUMEVIEWER_GRADIENT32 1
#define VOLUMEVIEWER_GRADIENT54 1
```

Compiling takes a very long time.

Calculating smooth gradients for light interactions takes a long time to compile for all the different shader variants (about an hour on our machines). If you don't need smooth gradients, you can prevent the 32 and 54 fetches gradient-functions from being compiled by commenting out one or both of these lines in Assets/VolumeViewerPro/shaders/

```
#define VOLUMEVIEWER_GRADIENT32 1
#define VOLUMEVIEWER_GRADIENT54 1
```

On our machines, commenting out both of these lines brings compile time down to 4 minutes. Since shader variants are cached by Unity, you only need to wait that long the first time you compile.

Files can't be loaded.

If you're having problems loading your files, try converting them to a different format. We recommend the free (public domain) tool ImageJ including the following plugins for DICOM and NIfTI support. Plugins need to be saved to the plugins folder of ImageJ.

https://imagej.nih.gov/ij/download.html
https://imagej.nih.gov/ij/plugins/import-dicom.html
https://imagej.nih.gov/ij/plugins/nifti.html

Data slices are out of order.

When a data set is split up into multiple files, the order is determined by the header of the file or the file names. A save naming convention to ensure the correct order for PNG and JPG files (which don't have information about their order in their headers) is a constant prefix followed by a constant number of digits representing the order. E.g. XYZ001.png, XYZ002.png ... XYZ128.png.

RayOffset doesn't seem to work correctly.

In order to reduce wood grain artifacts, a noise texture can be used to offset individual rays. We recommend setting FilterMode to point on the noise texture and not to generate Mip Maps.

How to save procedurally generated volumetric data to a Texture3D asset, so it will load much faster than by generating the data during runtime?

We created an example to answer this question. Please have a look at HowToMakeATexture3DAsset.cs in Assets/VolumeViewerPro/examples/scripts/. It should contain everything you need to know.