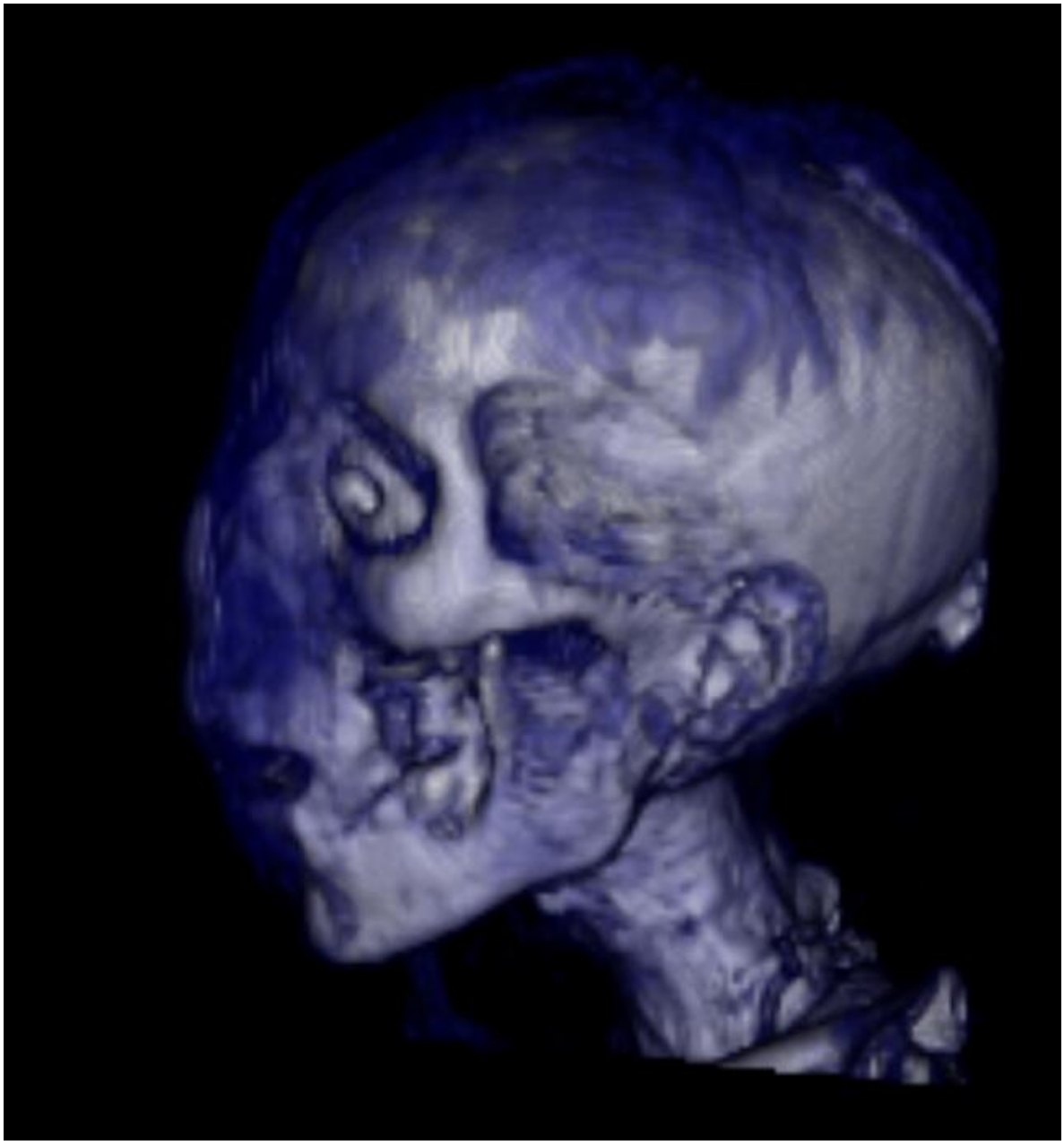
vol_ren.tcl

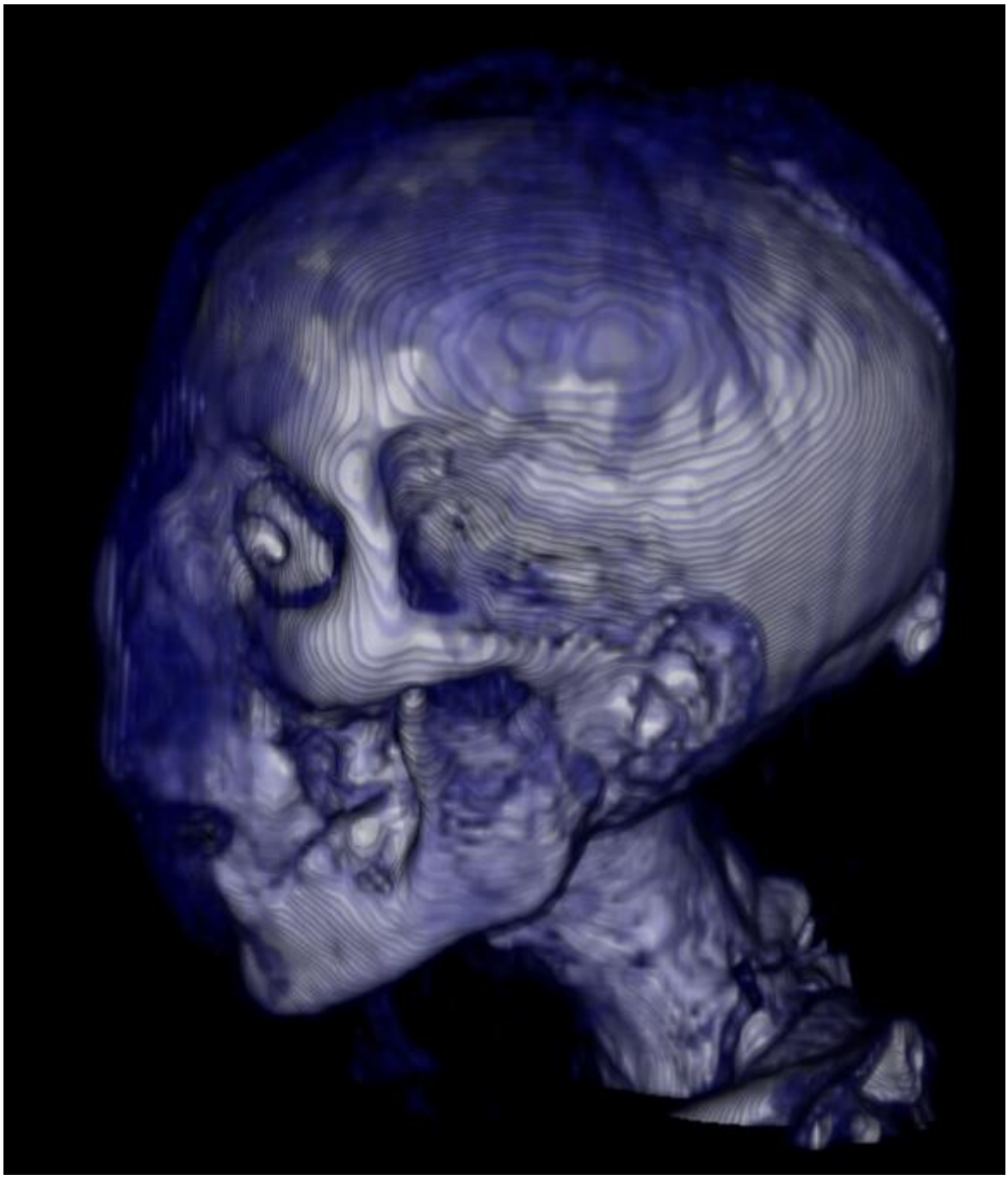
Picture

Initially

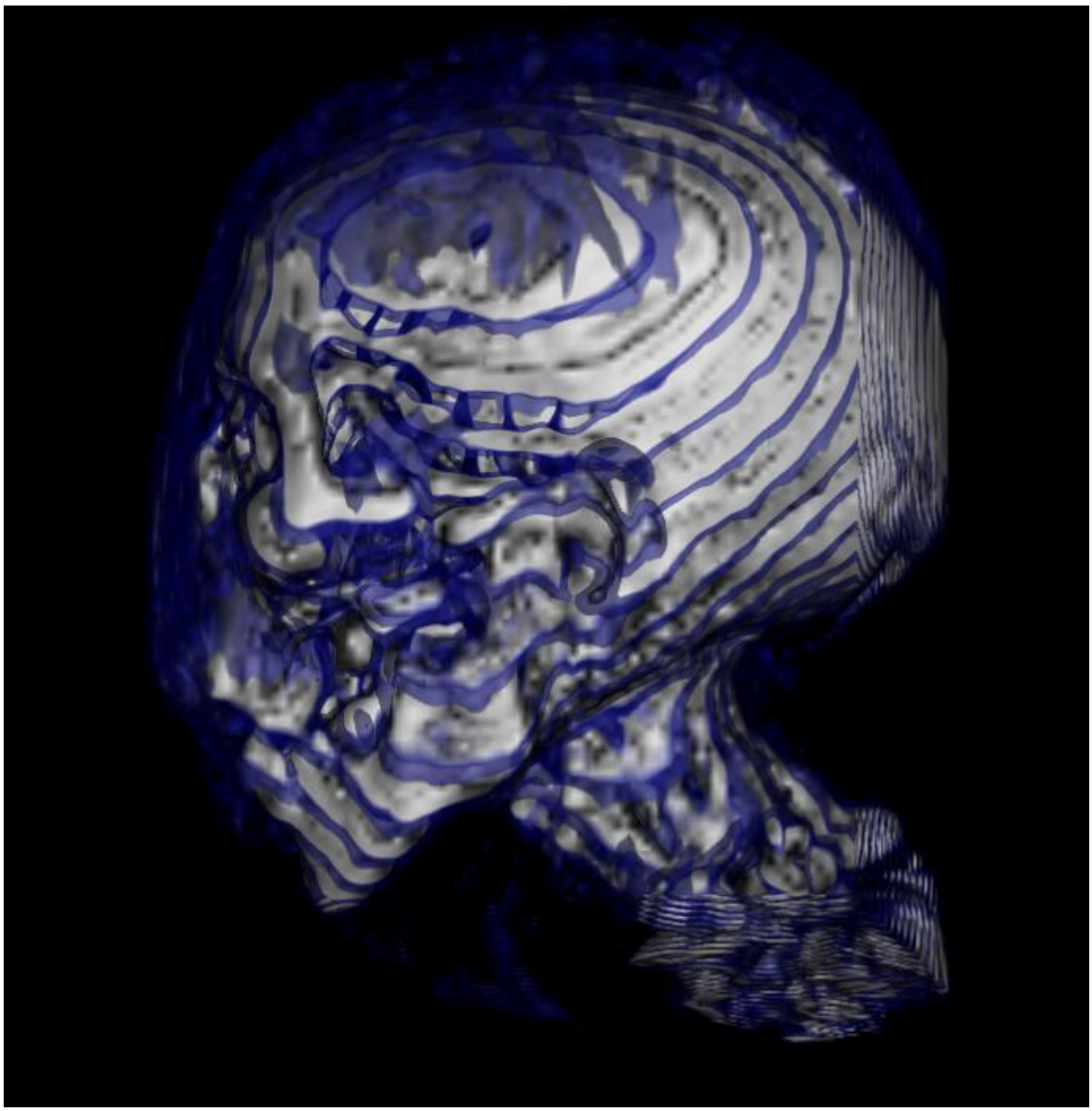


Try different sample distance spacing.

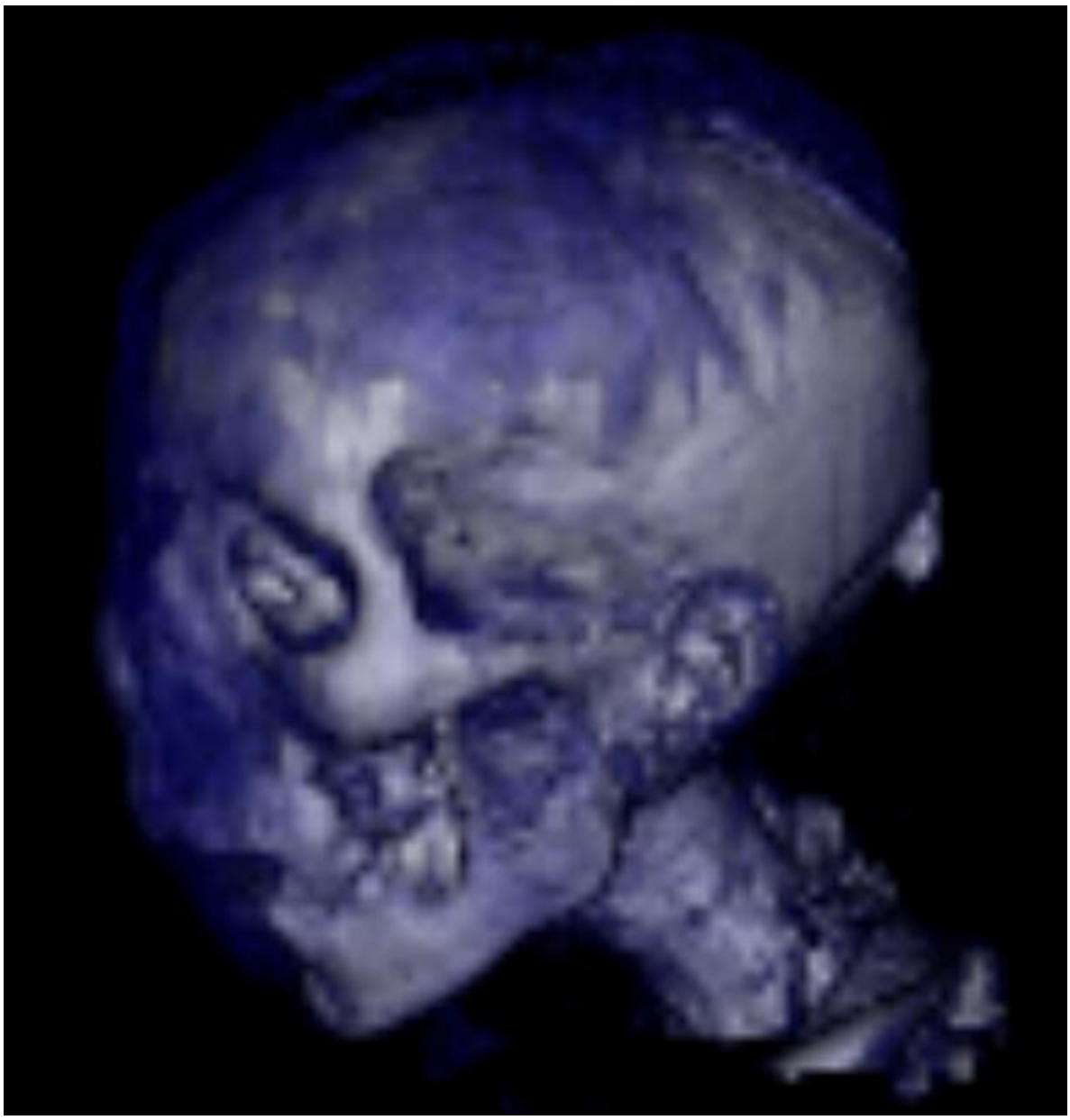
```
volumeMapper SetSampleDistance 1.0
```



```
volumeMapper SetSampleDistance 8.0
```

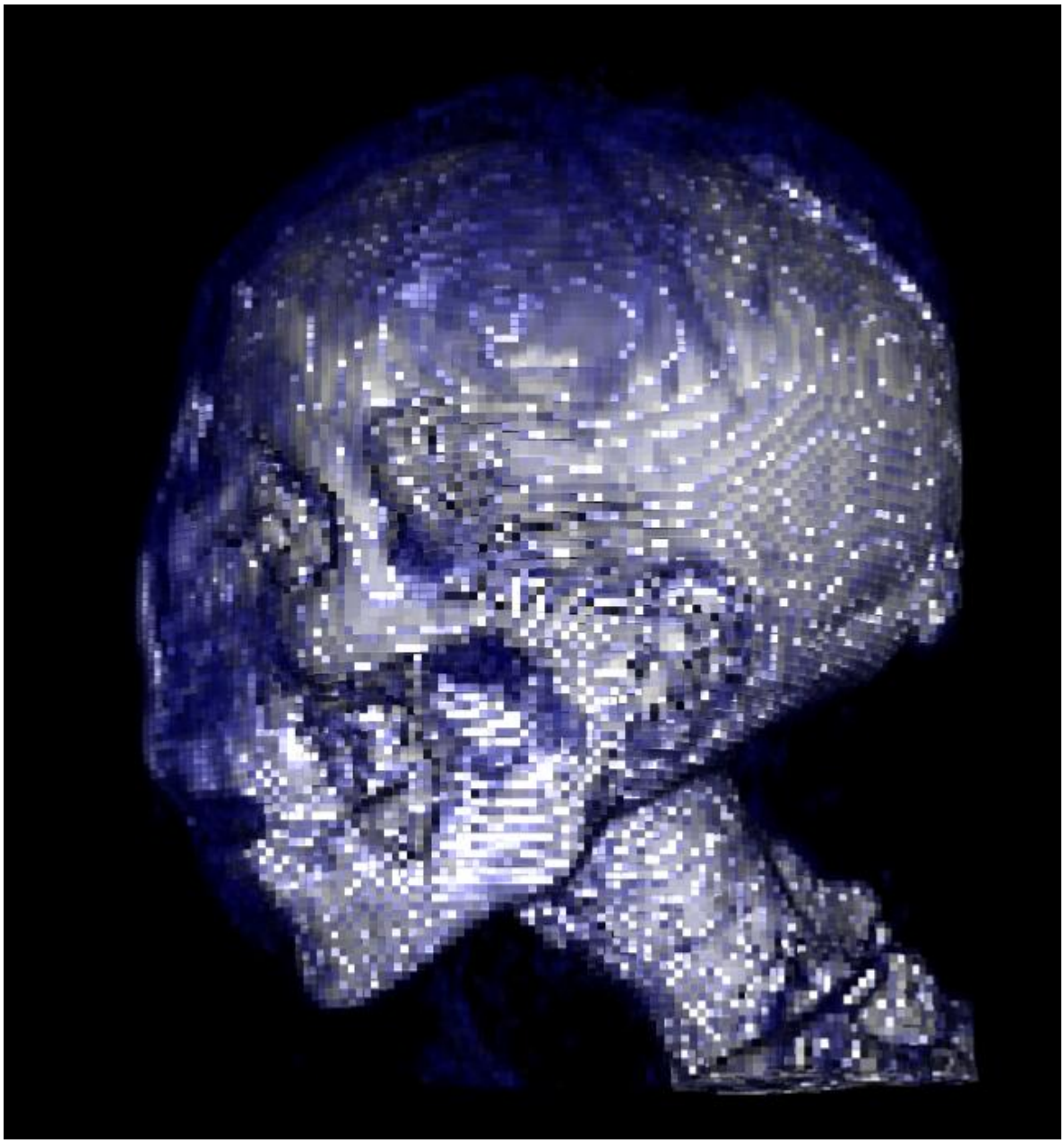


```
volumeMapper SetSampleDistance 0.2
```

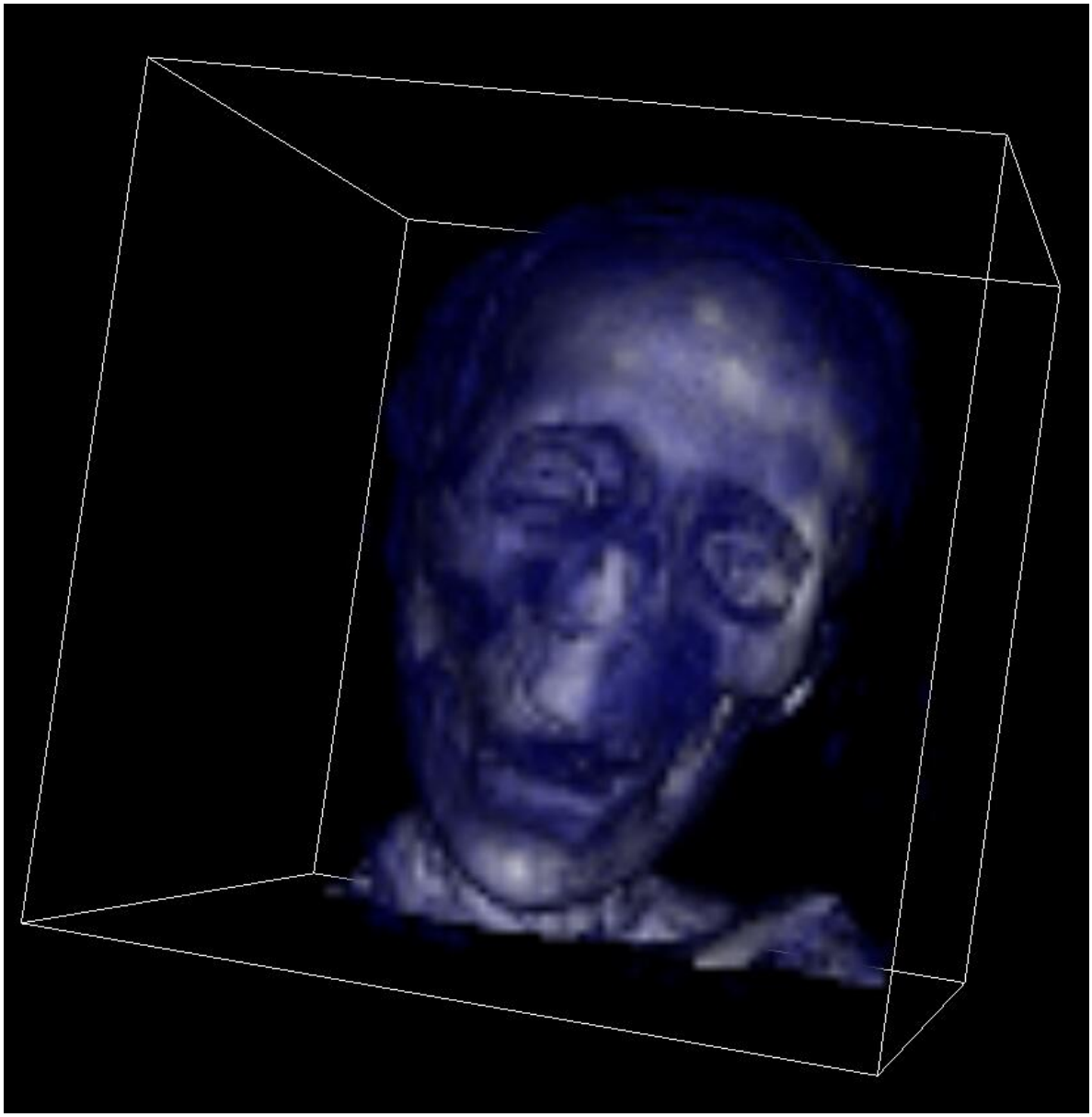


Try different interpolation type.

```
# Create properties, mappers, volume actors, and ray cast function
vtkVolumeProperty volumeProperty
    volumeProperty SetColor colorTransferFunction
    volumeProperty SetScalarOpacity opacityTransferFunction
    volumeProperty ShadeOn
    #volumeProperty SetInterpolationTypeToLinear
    volumeProperty SetInterpolationTypeToNearest
```

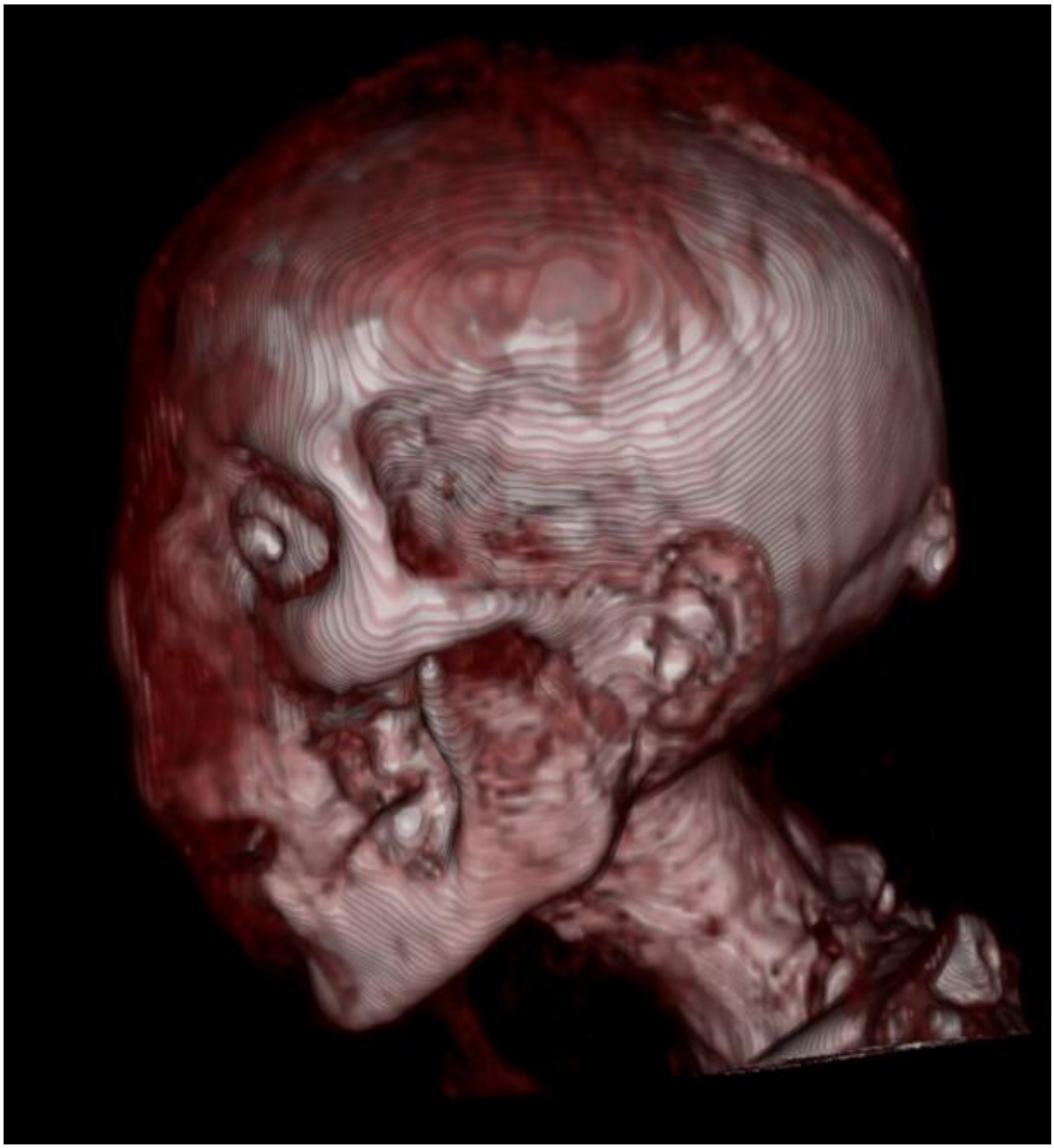


```
ren1 AddActor outlineActor
ren1 AddVolume volume
ren1 SetBackground 0 0 0
renWin Render
```



Change the color

```
vtkColorTransferFunction colorTransferFunction
colorTransferFunction AddRGBPoint 0 0.0 0.0 0.0
colorTransferFunction AddRGBPoint 50 0.0 0.0 0.0
colorTransferFunction AddRGBPoint 55 0.6 0.0 0.0
colorTransferFunction AddRGBPoint 80 0.6 0.0 0.0
colorTransferFunction AddRGBPoint 100 1.0 1.0 1.0
colorTransferFunction AddRGBPoint 120 1.0 1.0 1.0
```

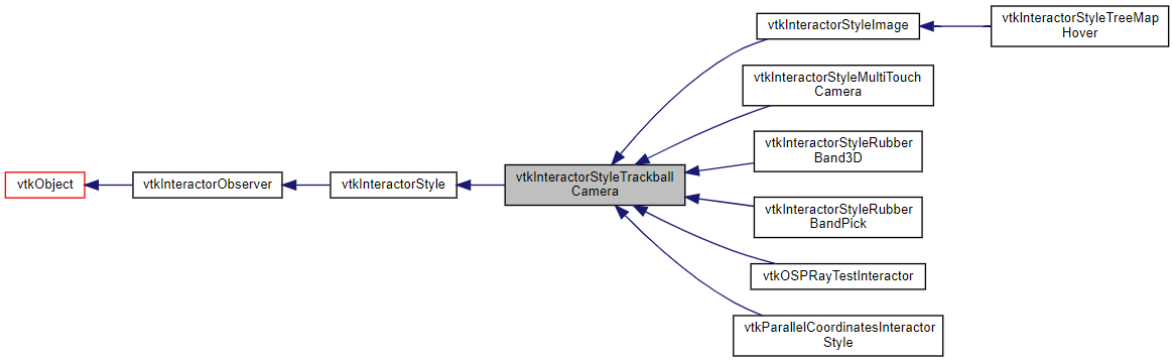


Code

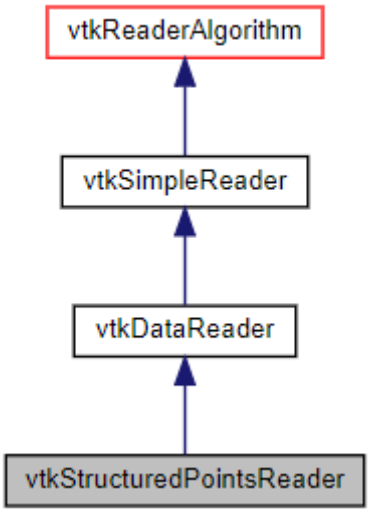
```
# Add this line and one after the renderer
vtkInteractorStyleTrackballCamera style


vtkStructuredPointsReader reader1
        reader1 SetFileName mummy.128.vtk
```

vtkInteractorStyleTrackballCamera allows the user to interactively manipulate (rotate, pan, etc.) the camera, the viewpoint of the scene. In trackball interaction, the magnitude of the mouse motion is proportional to the camera motion associated with a particular mouse binding.

For example, small left-button motions cause small changes in the rotation of the camera around its focal point. For a 3-button mouse, the left button is for rotation, the right button for zooming, the middle button for panning, and ctrl + left button for spinning. (With fewer mouse buttons, ctrl + shift + left button is for zooming, and shift + left button is for panning.)



vtkStructuredPointsReader is a source object that reads ASCII or binary structured points data files in vtk format (see text for format details). The output of this reader is a single **vtkStructuredPoints** data object. The superclass of this class, **vtkDataReader**, provides many methods for controlling the reading of the data file.




```
# Create transfer functions for opacity and color

vtkPiecewiseFunction opacityTransferFunction
```

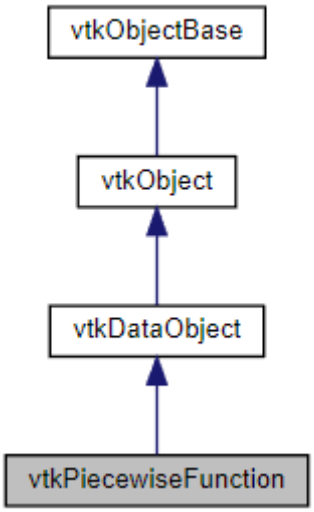
```
    opacityTransferFunction AddPoint 0 0.0
    opacityTransferFunction AddPoint 50 0.0
    opacityTransferFunction AddPoint 55 0.1
    opacityTransferFunction AddPoint 80 0.1
    opacityTransferFunction AddPoint 100 1.0
    opacityTransferFunction AddPoint 120 1.0

vtkColorTransferFunction colorTransferFunction
colorTransferFunction AddRGBPoint 0 0.0 0.0 0.0
colorTransferFunction AddRGBPoint 50 0.0 0.0 0.0
colorTransferFunction AddRGBPoint 55 0.0 0.0 0.6
colorTransferFunction AddRGBPoint 80 0.0 0.0 0.6
colorTransferFunction AddRGBPoint 100 1.0 1.0 1.0
colorTransferFunction AddRGBPoint 120 1.0 1.0 1.0
```

vtkPiecewiseFunction defines a piecewise function mapping. This mapping allows the addition of control points, and allows the user to control the function between the control points. A piecewise hermite curve is used between control points, based on the sharpness and midpoint parameters. A sharpness of 0 yields a piecewise linear function and a sharpness of 1 yields a piecewise constant function. The midpoint is the normalized distance between control points at which the curve reaches the median Y value. The midpoint and sharpness values specified when adding a node are used to control the transition to the next node (the last node's values are ignored) Outside the range of nodes, the values are 0 if Clamping is off, or the nearest node point if Clamping is on. Using the legacy methods for adding points (which do not have Sharpness and Midpoint parameters) will default to Midpoint = 0.5 (halfway between the control points) and Sharpness = 0.0 (linear).

int vtkPiecewiseFunction::AddPoint (double x, double y)

Add/Remove points to/from the function.

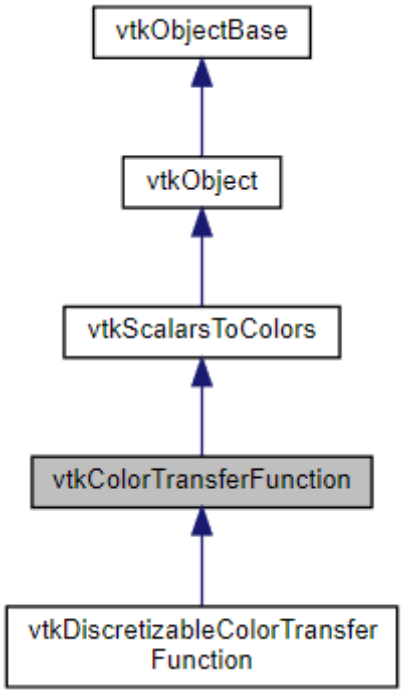
If a duplicate point is added then the function value is changed at that location. Return the index of the point (0 based), or -1 on error.



vtkColorTransferFunction is a color mapping in RGB or HSV space that uses piecewise hermite functions to allow interpolation that can be piecewise constant, piecewise linear, or somewhere in-between (a modified piecewise hermite function that squishes the function according to a sharpness parameter). The function also allows for the specification of the midpoint (the place where the function reaches the average of the two bounding nodes) as a normalize distance between nodes.

int vtkColorTransferFunction::AddRGBPoint (double x, double r, double g, double b)

Add/Remove a point to/from the function defined in RGB or HSV Return the index of the point (0 based), or -1 on error.




```
# Create properties, mappers, volume actors, and ray cast function
vtkVolumeProperty volumeProperty
    volumeProperty SetColor colorTransferFunction
    volumeProperty SetScalarOpacity opacityTransferFunction
    volumeProperty ShadeOn
    volumeProperty SetInterpolationTypeToLinear
    #volumeProperty SetInterpolationTypeToNearest

vtkVolumeRayCastCompositeFunction  compositeFunction

vtkVolumeRayCastMapper volumeMapper
    volumeMapper SetInput [reader1 GetOutput]
    volumeMapper SetVolumeRayCastFunction compositeFunction
    volumeMapper SetSampleDistance 0.5
    #volumeMapper SetSampleDistance 1.0
    #volumeMapper SetSampleDistance 8.0
    #volumeMapper SetSampleDistance 0.2

vtkVolume volume
    volume SetMapper volumeMapper
    volume SetProperty volumeProperty
```

vtkVolumeProperty is used to represent common properties associated with volume rendering. This includes properties for determining the type of interpolation to use when sampling a volume, the color of a volume, the scalar opacity of a volume, the gradient opacity of a volume, and the shading parameters of a volume.

void vtkVolumeProperty::SetInterpolationTypeToLinear ( )

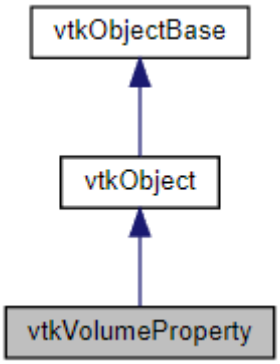
Set the interpolation type for sampling a volume.

Initial value is VTK_NEAREST_INTERPOLATION.

void vtkVolumeProperty::SetInterpolationTypeToNearest()

Set the interpolation type for sampling a volume.

Initial value is VTK_NEAREST_INTERPOLATION.



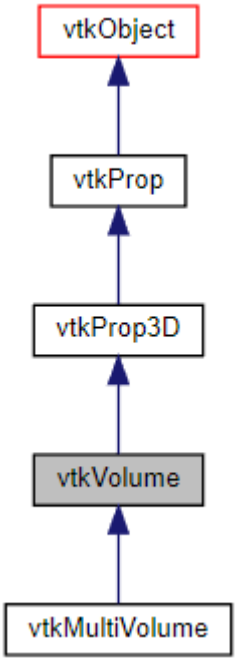
vtkVolume is used to represent a volumetric entity in a rendering scene. It inherits functions related to the volume's position, orientation and origin from vtkProp3D. The volume maintains a reference to the volumetric data (i.e., the volume mapper). The volume also contains a reference to a volume property which contains all common volume rendering parameters.

void vtkVolume::SetMapper (vtkAbstractVolumeMapper * mapper)

Set/Get the volume mapper.

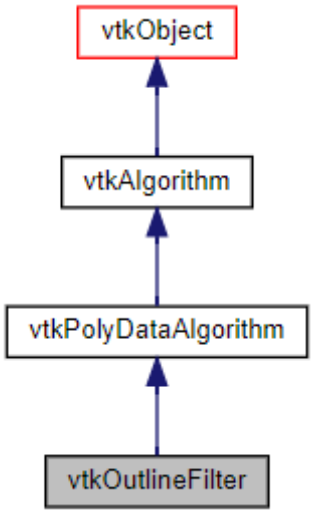
virtual void vtkVolume::SetProperty (vtkVolumeProperty * property)

Set/Get the volume property.

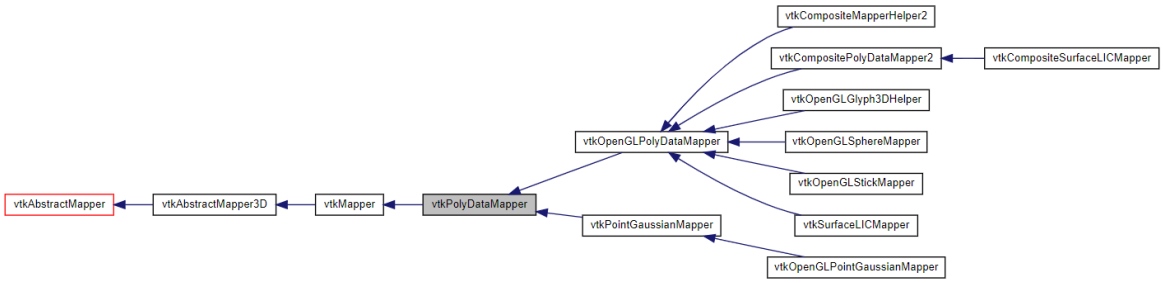


```
# Create outline
vtkOutlineFilter outline
outline SetInput [reader1 GetOutput]

vtkPolyDataMapper outlineMapper
outlineMapper SetInput [outline GetOutput]

vtkActor outlineActor
    outlineActor SetMapper outlineMapper
eval [outlineActor GetProperty] SetColor 1 1 1
```

vtkOutlineFilter is a filter that generates a wireframe outline of any dataset or composite dataset. An outline consists of the twelve edges of the dataset bounding box. An option exists for generating faces instead of a wireframe outline.



vtkPolyDataMapper is a class that maps polygonal data (i.e., **vtkPolyData**) to graphics primitives. **vtkPolyDataMapper** serves as a superclass for device-specific poly data mappers, that actually do the mapping to the rendering/graphics hardware/software.

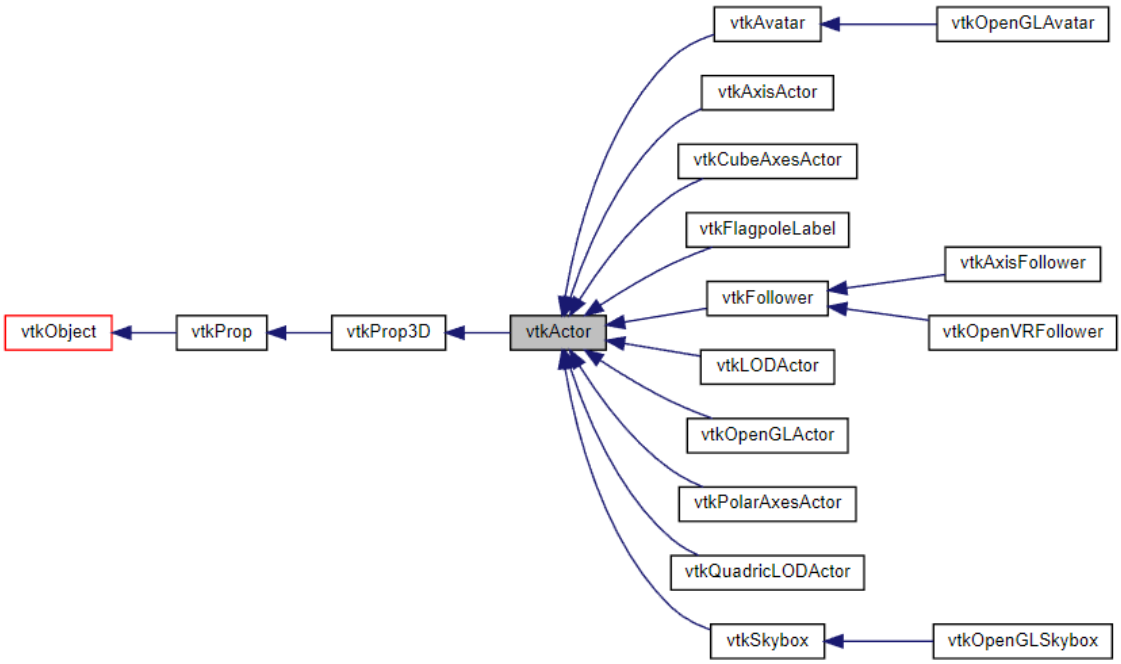


vtkActor is used to represent an entity in a rendering scene. It inherits functions related to the actors position, and orientation from **vtkProp**. The actor also has scaling and maintains a reference to the defining geometry (i.e., the mapper), rendering properties, and possibly a texture map. **vtkActor** combines these instance variables into one 4x4 transformation matrix as follows: [x y z 1] = [x y z 1] Translate(-origin) Scale(scale) Rot(y) Rot(x) Rot (z) Trans(origin) Trans(position)

virtual void vtkActor::SetMapper (vtkMapper *)

This is the method that is used to connect an actor to the end of a visualization pipeline, i.e.

the mapper. This should be a subclass of vtkMapper. Typically vtkPolyDataMapper and vtkDataSetMapper will be used.




```
# Okay now the graphics stuff
vtkRenderer ren1
vtkRenderWindow renWin
    renWin AddRenderer ren1
    renWin SetSize 512 512

vtkRenderWindowInteractor iren
    iren SetRenderWindow renWin
iren SetInteractorStyle style

#ren1 AddActor outlineActor
ren1 AddVolume volume
ren1 SetBackground 0 0 0
renWin Render
```

vtkRenderer provides an abstract specification for renderers. A renderer is an object that controls the rendering process for objects. Rendering is the process of converting geometry, a specification for lights, and a camera view into an image. **vtkRenderer** also performs coordinate transformation between world coordinates, view coordinates (the computer graphics rendering coordinate system), and display coordinates (the actual screen coordinates on the display device). Certain advanced rendering features such as two-sided lighting can also be controlled.

void vtkRenderer::AddActor (vtkProp * p)

Add/Remove different types of props to the renderer.

These methods are all synonyms to AddViewProp and RemoveViewProp. They are here for convenience and backwards compatibility.

void vtkRenderer::AddVolume (vtkProp * p)

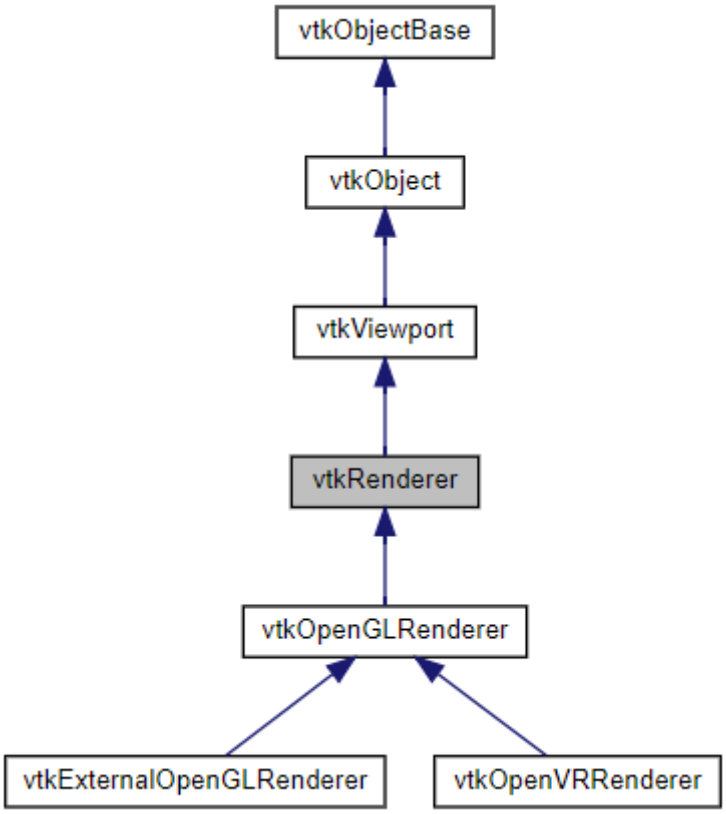
Add/Remove different types of props to the renderer.

These methods are all synonyms to AddViewProp and RemoveViewProp. They are here for convenience and backwards compatibility.

virtual void vtkRenderer::SetBackgroundTexture (vtkTexture *)

Set/Get the texture to be used for the monocular or stereo left eye background.

If set and enabled this gets the priority over the gradient background.

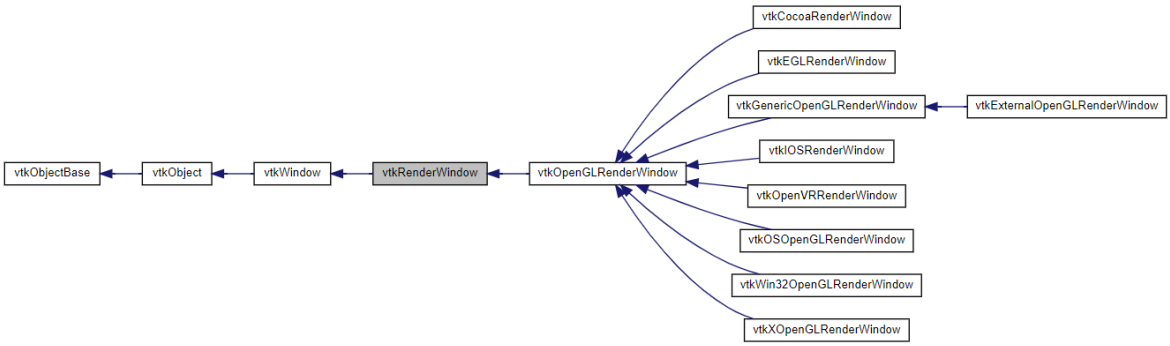


vtkRenderWindow is an abstract object to specify the behavior of a rendering window. A rendering window is a window in a graphical user interface where renderers draw their images. Methods are provided to synchronize the rendering process, set window size, and control double buffering. The window also allows rendering in stereo. The interlaced render stereo type is for output to a VRex stereo projector. All of the odd horizontal lines are from the left eye, and the even lines are from the right eye. The user has to make the render window aligned with the VRex projector, or the eye will be swapped.

virtual void vtkRenderWindow::AddRenderer (vtkRenderer *)

Add a renderer to the list of renderers.

Reimplemented in **vtkOpenVRRenderWindow**.



vtkRenderWindowInteractor provides a platform-independent interaction mechanism for mouse/key/time events. It serves as a base class for platform-dependent implementations that handle routing of mouse/key/timer messages to **vtkInteractorObserver** and its subclasses. **vtkRenderWindowInteractor** also provides controls for picking, rendering frame rate, and headlights.

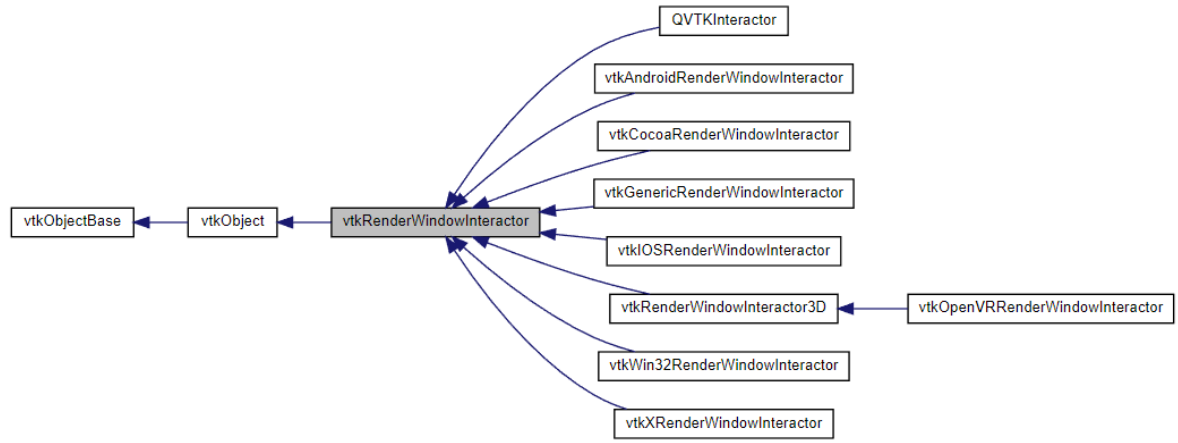
void vtkRenderWindowInteractor::SetRenderWindow (vtkRenderWindow * aren)

Set/Get the rendering window being controlled by this object.

virtual void vtkRenderWindowInteractor::SetInteractorStyle (vtkInteractorObserver *)

External switching between joystick/trackball/new? modes.

Initial value is a **vtkInteractorStyleSwitch** object.



vol_mip.tcl

Initially

Try different sample distance spacing.

```
volumeMapper SetSampleDistance 1.0
```

```
volumeMapper SetSampleDistance 8.0
```

```
volumeMapper SetSampleDistance 0.2
```