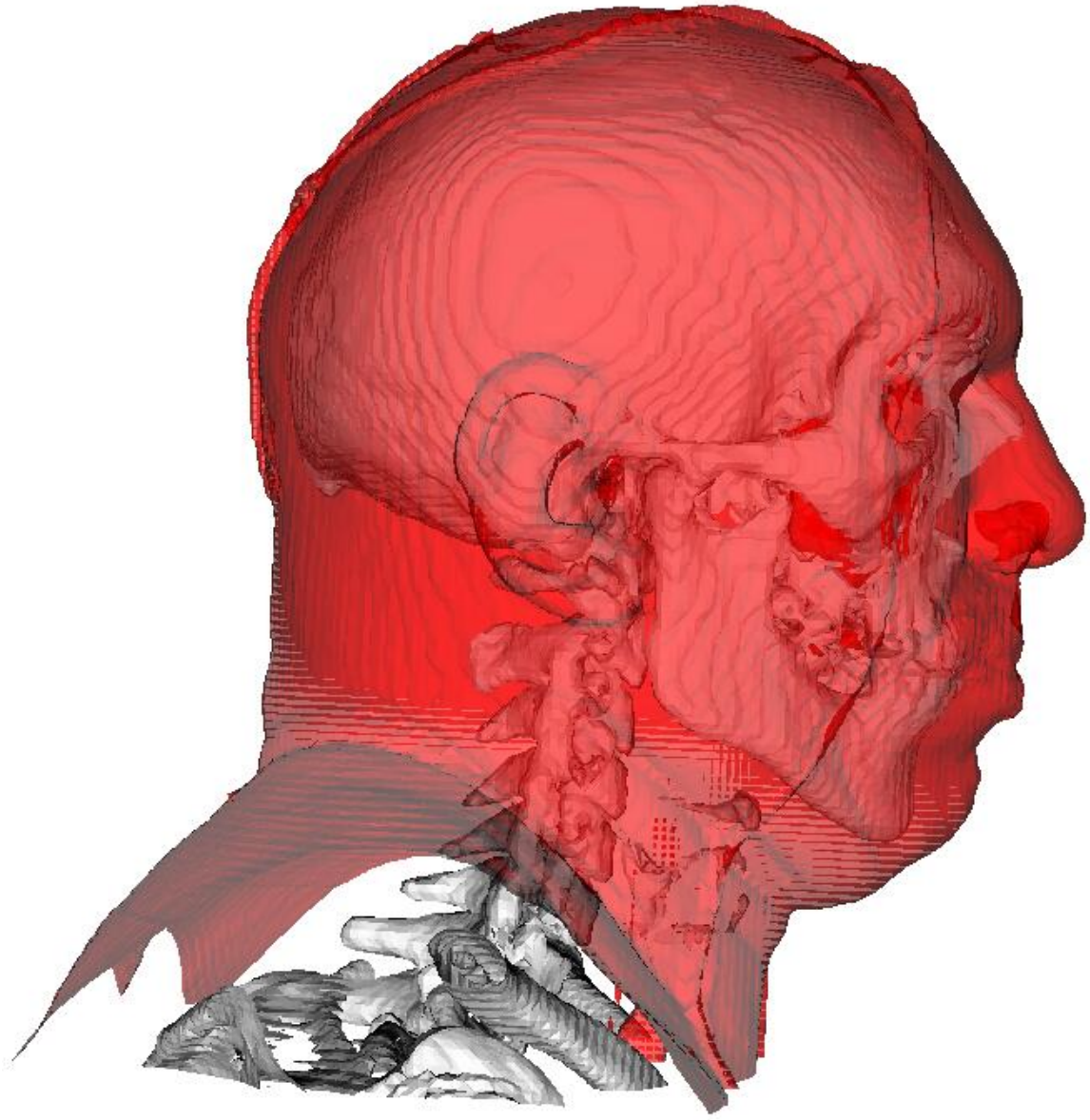
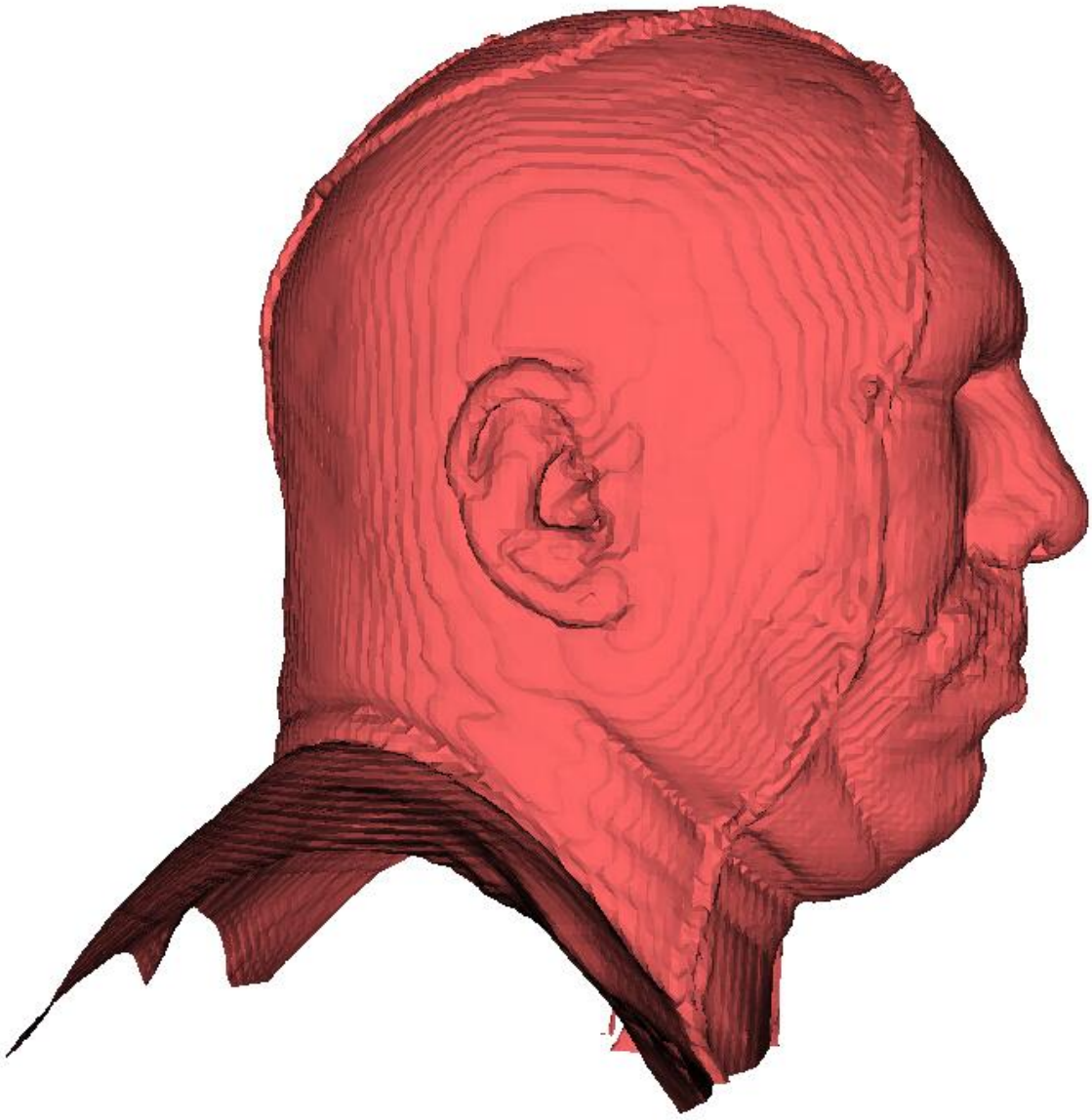
Picture

Initially



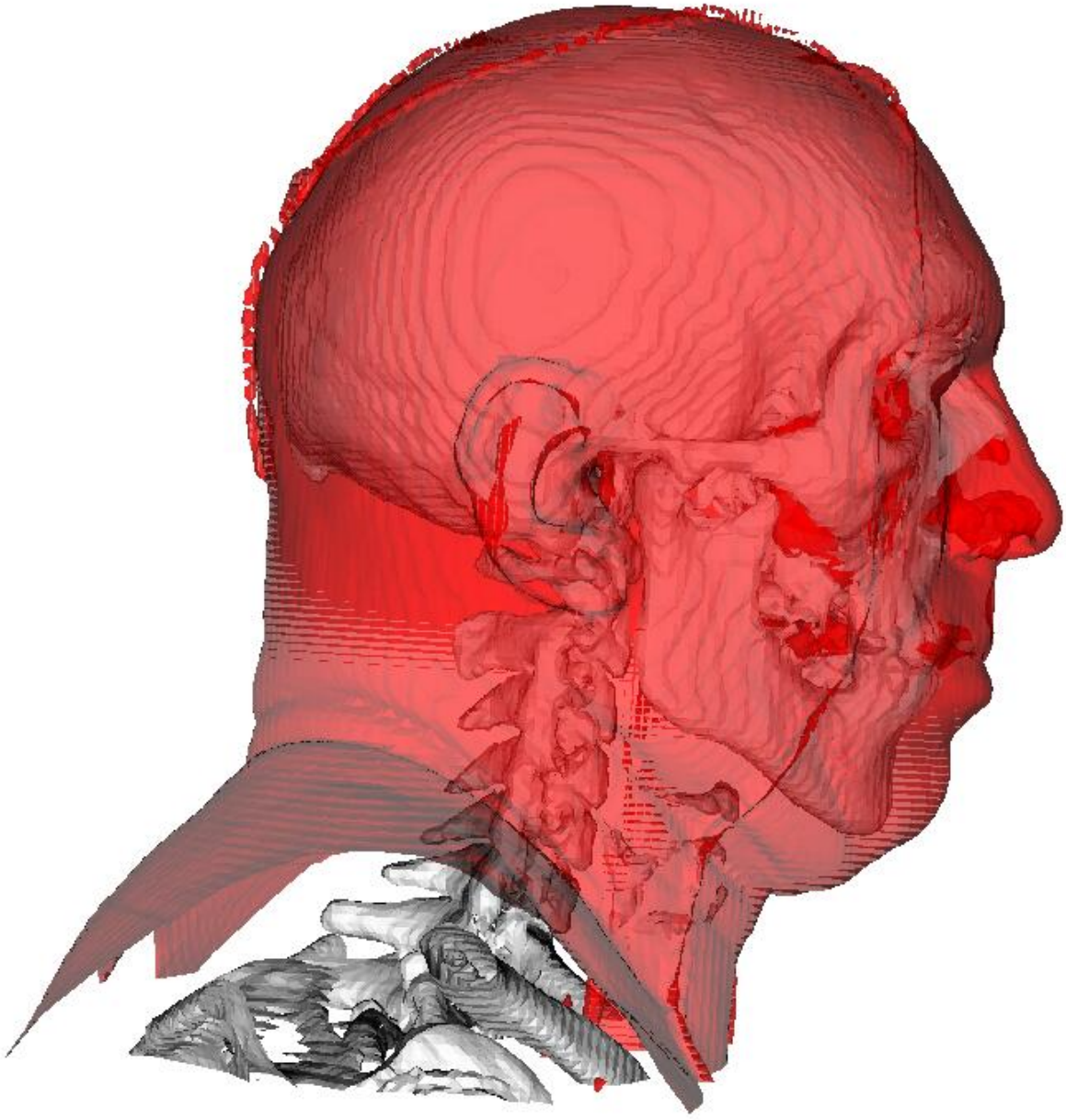
Try different contour_bone values.

```
########################## Contour Filter for Head Bone ######################

vtkContourFilter contour_bone
    contour_bone SetInputConnection [reader1 GetOutputPort]
    contour_bone SetNumberOfContours 1
    contour_bone SetValue 0 25.0
```

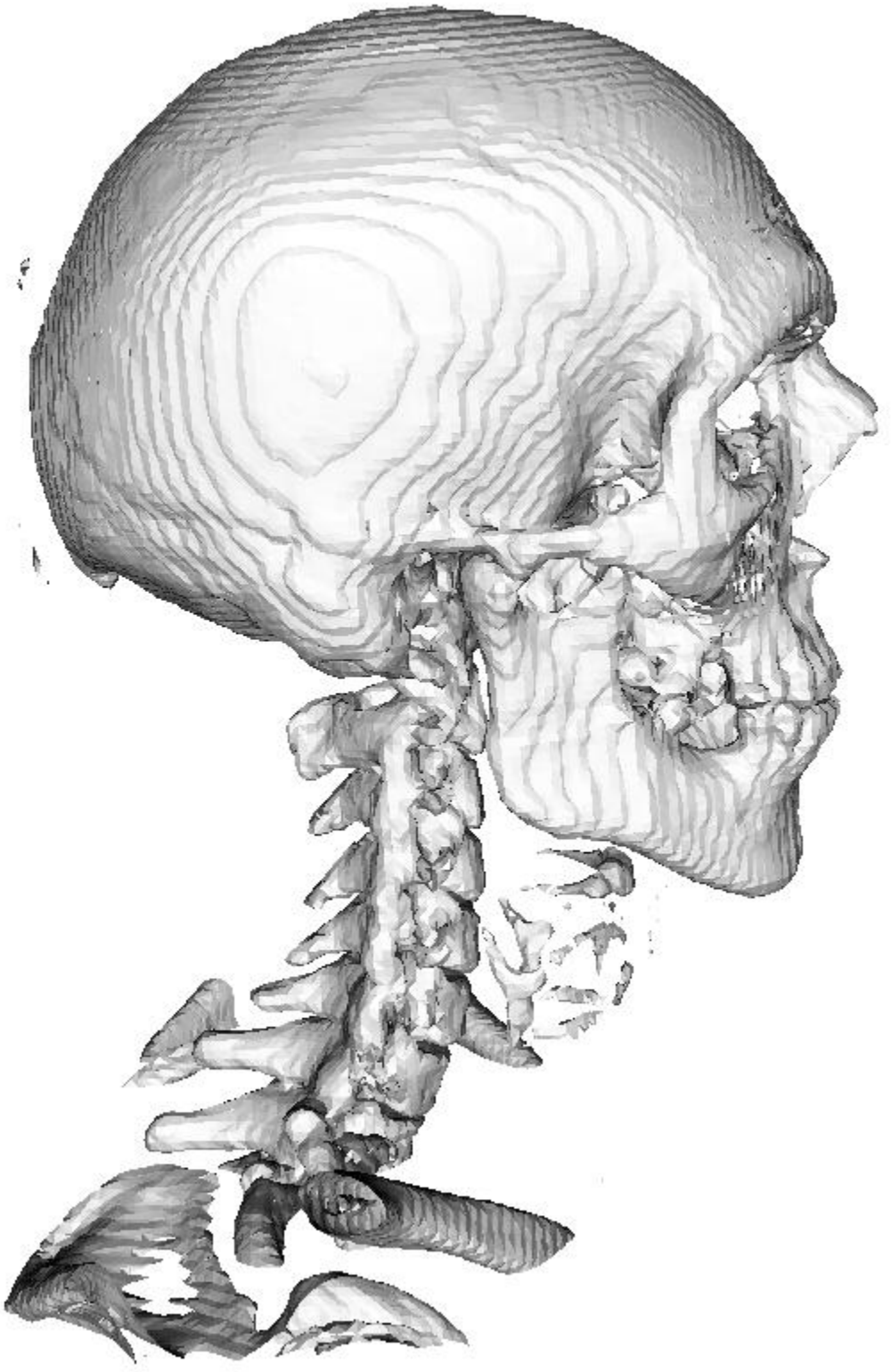


Try different contour_skin values.

```
########################## Contour Filter for Head Skin ######################

vtkContourFilter contour_skin
    contour_skin SetInputConnection [reader1 GetOutputPort]
    contour_skin SetNumberOfContours 1
    contour_skin SetValue 0 50.0
```

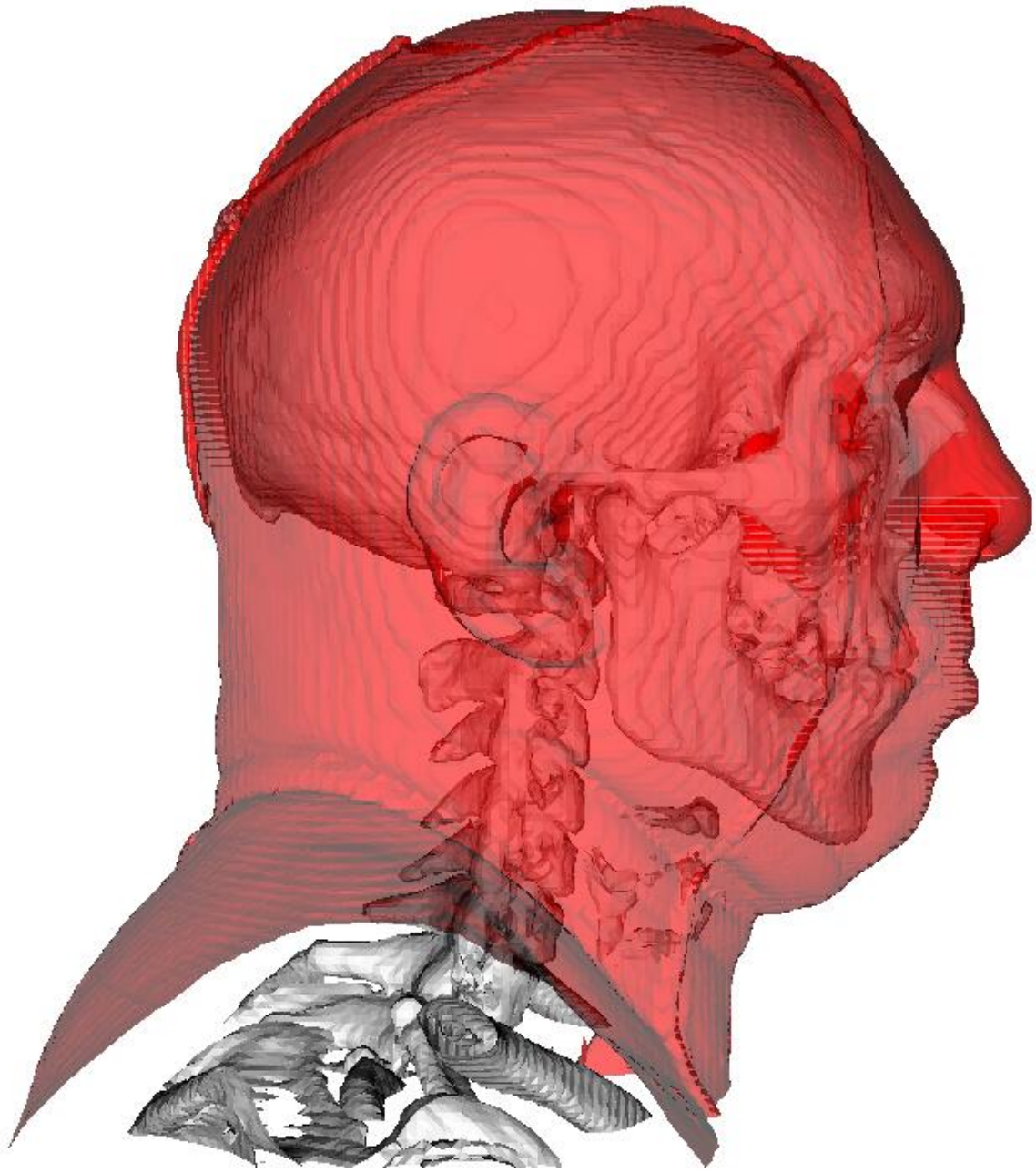


```
########################## Contour Filter for Head Skin ######################

vtkContourFilter contour_skin
    contour_skin SetInputConnection [reader1 GetOutputPort]
    contour_skin SetNumberOfContours 1
    contour_skin SetValue 0 0.0
```



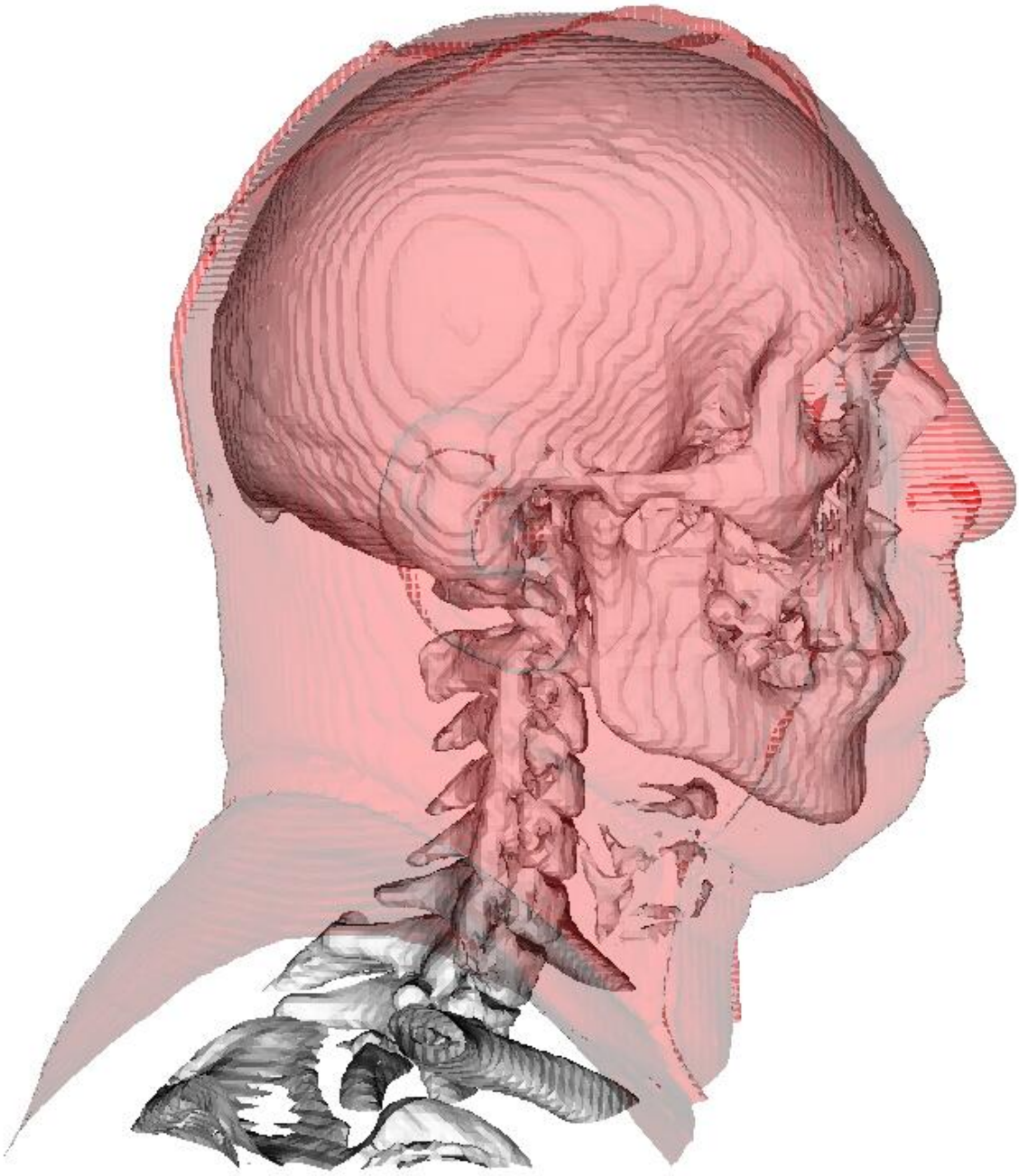
Try different Skin_NumberOfContours

```
########################## Contour Filter for Head Skin ######################

vtkContourFilter contour_skin
    contour_skin SetInputConnection [reader1 GetOutputPort]
    contour_skin SetNumberOfContours 30
    contour_skin SetValue 0 25.0
```



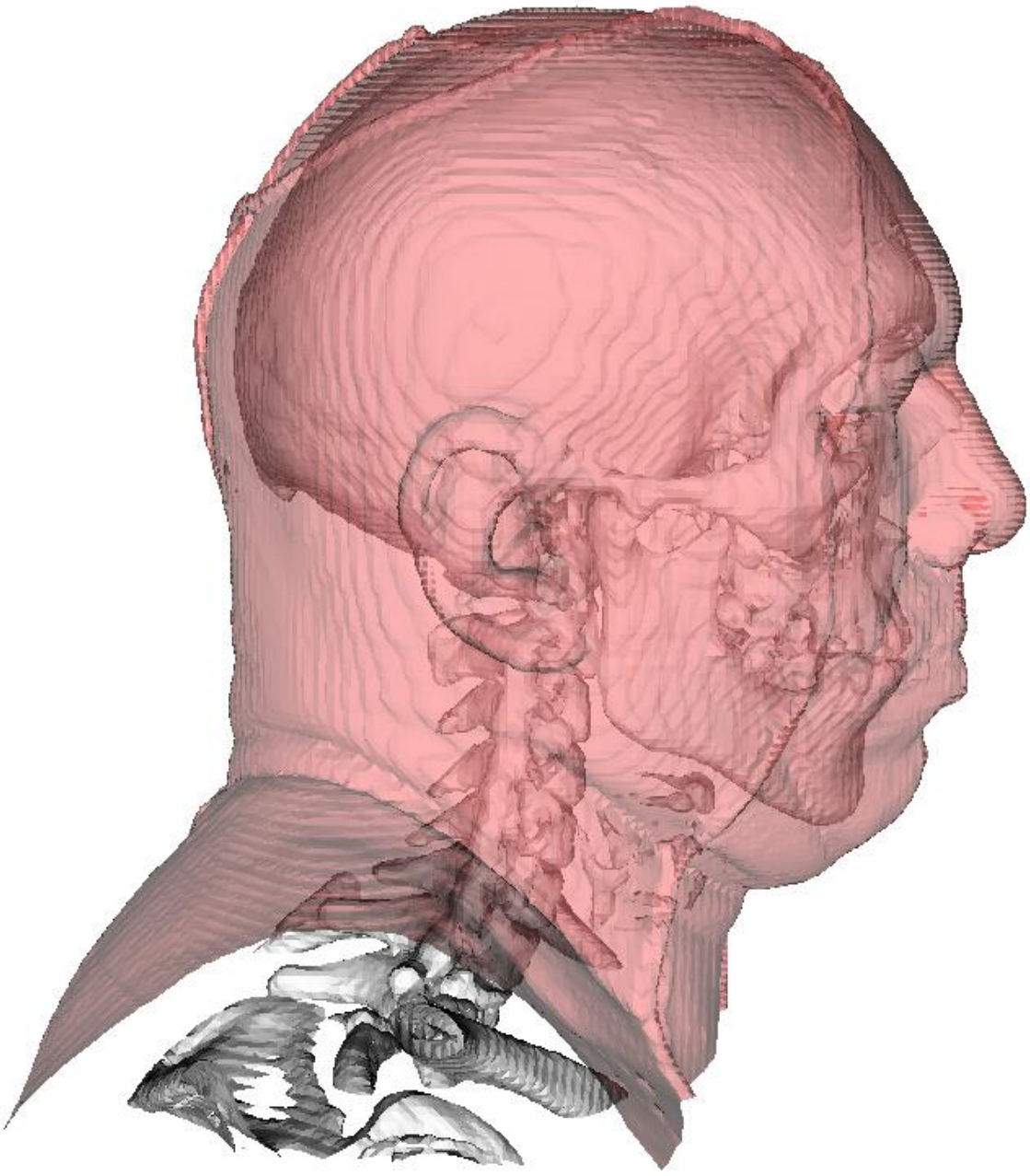
Try different Skin_AlphaRange

```
############################## Look up Table Skin ##############################

vtkLookupTable lut_skin
    lut_skin SetNumberOfColors 256
    lut_skin SetTableRange 0 255

lut_skin SetHueRange 0.0 0.0
lut_skin SetSaturationRange 0.0  1.0
lut_skin SetValueRange 0.0 1.0
lut_skin SetAlphaRange 0.3 0.3
lut_skin Build
```



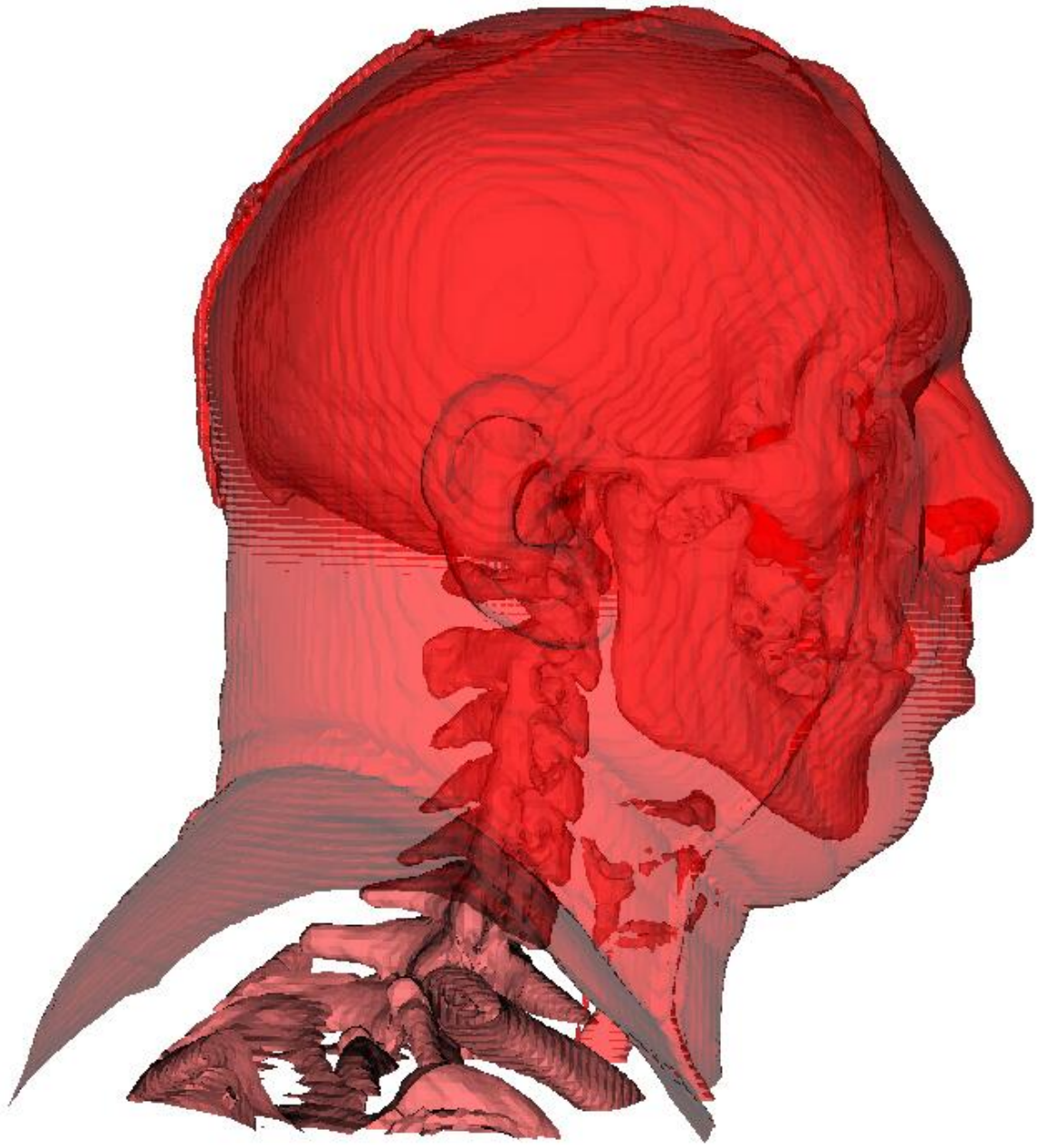
Try different Skin_SaturationRange

```
############################## Look up Table Skin ##############################

vtkLookupTable lut_skin
    lut_skin SetNumberOfColors 256
    lut_skin SetTableRange 0 255

lut_skin SetHueRange 0.0 0.0
lut_skin SetSaturationRange 0.0  0.5
lut_skin SetValueRange 0.0 1.0
lut_skin SetAlphaRange 0.6 0.6
lut_skin Build
```



Try different Bone_SaturationRange

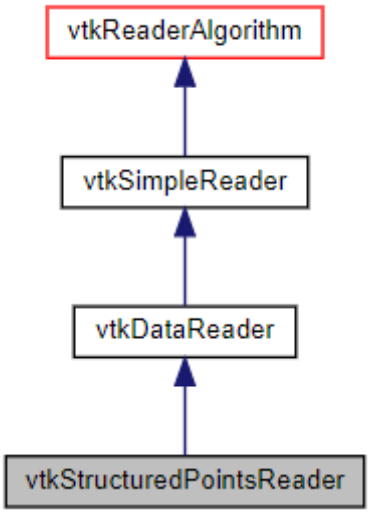
```
############################## Look up Table Bone ##############################

vtkLookupTable lut_bone
    lut_bone SetNumberOfColors 256
    lut_bone SetTableRange 0 255

lut_bone SetHueRange 0.0 1.0
lut_bone SetSaturationRange 0.0  0.5
lut_bone SetValueRange 0.0 1.0
lut_bone SetAlphaRange 1.0 1.0
lut_bone Build
```



Code

```
############################## Read the Head Data#########################

vtkStructuredPointsReader reader1
    reader1 SetFileName head.120.vtk
```

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.




```
############################## Look up Table Bone ##############################

vtkLookupTable lut_bone
    lut_bone SetNumberOfColors 256
    lut_bone SetTableRange 0 255

lut_bone SetHueRange 0.0 1.0
lut_bone SetSaturationRange 0.0  0.0
lut_bone SetValueRange 0.0 1.0
lut_bone SetAlphaRange 1.0 1.0
lut_bone Build
```

vtkLookupTable is an object that is used by mapper objects to map scalar values into RGBA (red-green-blue-alpha) color specification, or RGBA into scalar values. The color table can be created by direct insertion of color values, or by specifying a hue, saturation, value, and alpha range and generating a table.

virtual void vtkLookupTable::SetNumberOfColors (vtkIdType)

Set the number of colors in the lookup table.

Use *SetNumberOfTableValues()* instead, it can be used both before and after the table has been built whereas *SetNumberOfColors()* has no effect after the table has been built.

virtual void vtkLookupTable::SetTableRange (double min, double max)

Set/Get the minimum/maximum scalar values for scalar mapping.

Scalar values less than minimum range value are clamped to minimum range value. Scalar values greater than maximum range value are clamped to maximum range value.

The *TableRange* values are only used when *IndexedLookup* is false.

virtual void vtkLookupTable::SetHueRange (double , double)

Set the range in hue (using automatic generation).

Hue ranges between [0,1].

virtual void vtkLookupTable::SetSaturationRange (double , double)

Set the range in saturation (using automatic generation).

Saturation ranges between [0,1].

virtual void vtkLookupTable::SetValueRange (double , double)

Set the range in value (using automatic generation).

Value ranges between [0,1].

virtual void vtkLookupTable::SetAlphaRange (double , double)

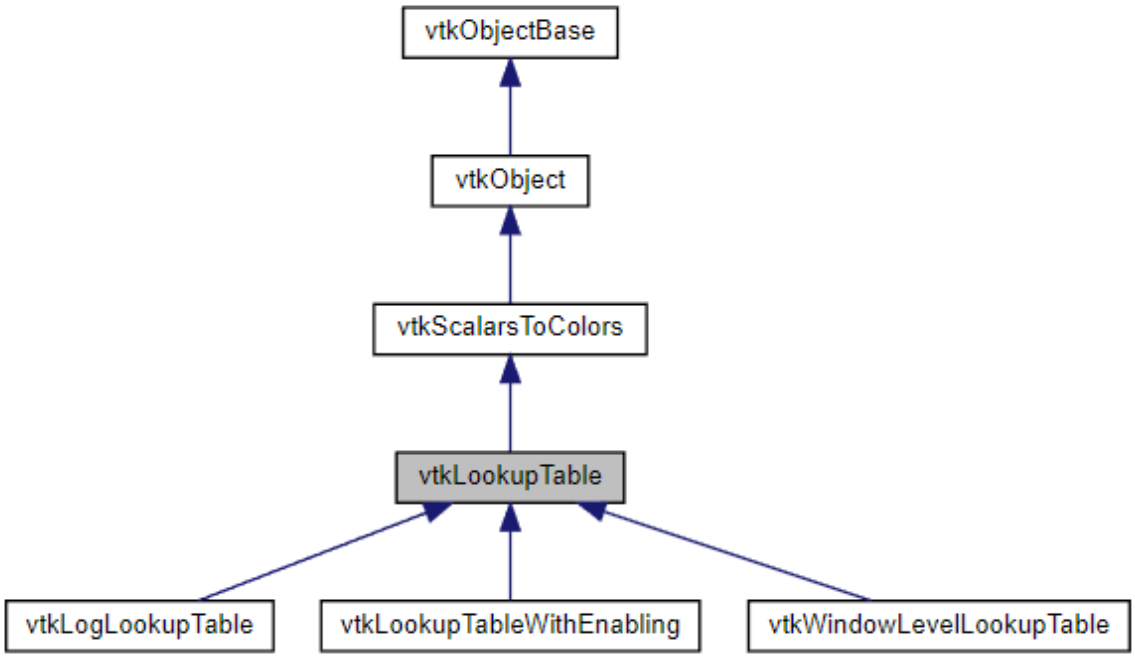
Set the range in alpha (using automatic generation).

Alpha ranges from [0,1].

void vtkLookupTable::Build ( )

Generate lookup table from hue, saturation, value, alpha min/max values.

Table is built from linear ramp of each value.

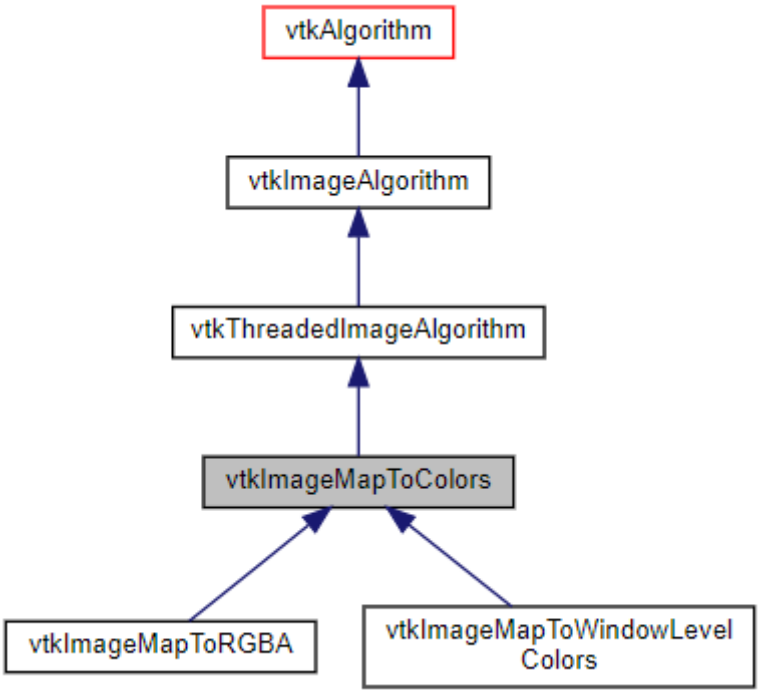



```
############################## ImageMapToColors Module Bone
###################
 vtkImageMapToColors iMtC_bone
    iMtC_bone SetInputConnection [reader1 GetOutputPort]
    iMtC_bone SetLookupTable lut_bone
```

The **vtkImageMapToColors** filter will take an input image of any valid scalar type, and map the first component of the image through a lookup table. The result is an image of type VTK_UNSIGNED_CHAR. If the lookup table is not set, or is set to nullptr, then the input data will be passed through if it is already of type VTK_UNSIGNED_CHAR.

virtual void vtkImageMapToColors::SetLookupTable (vtkScalarsToColors *)

Set the lookup table.




```
########################## Contour Filter for Head Bone ######################

vtkContourFilter contour_bone
    contour_bone SetInputConnection [reader1 GetOutputPort]
    contour_bone SetNumberOfContours 1
    contour_bone SetValue 0 75.0
```

vtkContourFilter is a filter that takes as input any dataset and generates on output isosurfaces and/or isolines. The exact form of the output depends upon the dimensionality of the input data. Data consisting of 3D cells will generate isosurfaces, data consisting of 2D cells will generate isolines, and data with 1D or 0D cells will generate isopoints. Combinations of output type are possible if the input dimension is mixed.

void vtkContourFilter::SetNumberOfContours (int number)

Set the number of contours to place into the list.

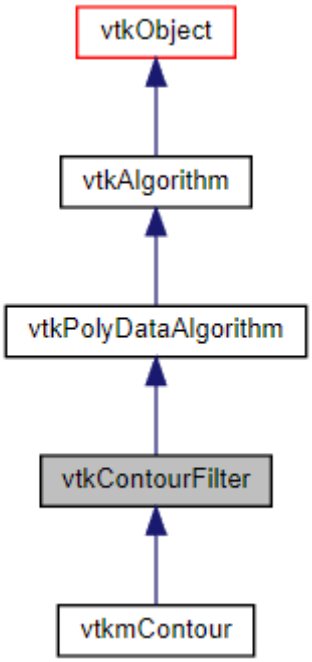
We only really need to use this method to reduce list size. The method *SetValue()* will automatically increase list size as needed.

void vtkContourFilter::SetValue (int i, double value)

Methods to set / get contour values.

Set a particular contour value at contour number i.

The index i ranges between 0<=i<NumberOfContours.



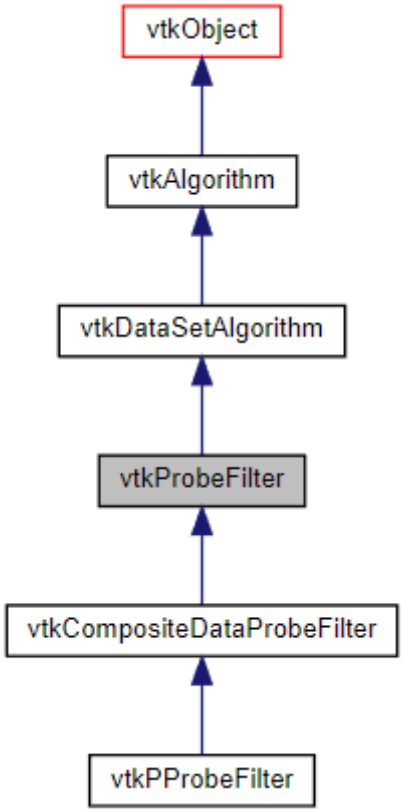

```
############################## Probe Filter for Bone ##########################

vtkProbeFilter probe_bone
    probe_bone SetInputConnection [contour_bone GetOutputPort]
    probe_bone SetSourceConnection [iMtC_bone GetOutputPort]
```

vtkProbeFilter is a filter that computes point attributes (e.g., scalars, vectors, etc.) at specified point positions. The filter has two inputs: the Input and Source. The Input geometric structure is passed through the filter. The point attributes are computed at the Input point positions by interpolating into the source data. For example, we can compute data values on a plane (plane specified as Input) from a volume (Source). The cell data of the source data is copied to the output based on in which source cell each input point is. If an array of the same name exists both in source's point and cell data, only the one from the point data is probed.

void vtkProbeFilter::SetSourceConnection (vtkAlgorithmOutput * algOutput)

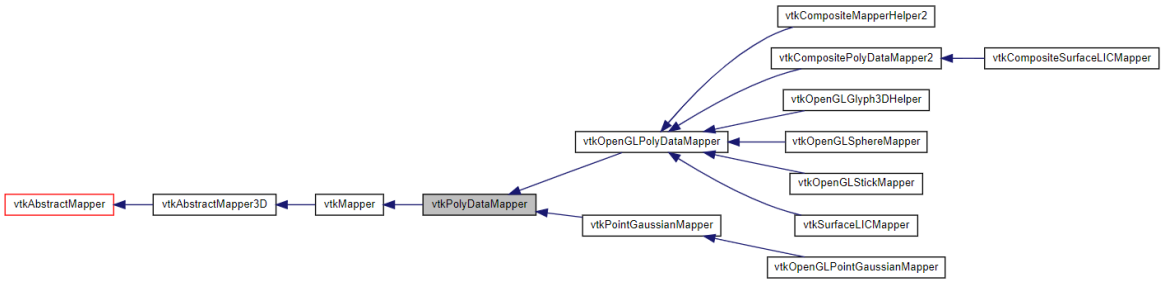
Specify the data set that will be probed at the input points.

The Input gives the geometry (the points and cells) for the output, while the Source is probed (interpolated) to generate the scalars, vectors, etc. for the output points based on the point locations.




```
############################## PolyData Mapper Bone
##############################

vtkPolyDataMapper mapper_bone
    mapper_bone SetInputConnection [probe_bone GetOutputPort]
        mapper_bone SetLookupTable lut_bone
        mapper_bone SetColorModeToMapScalars
```

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.



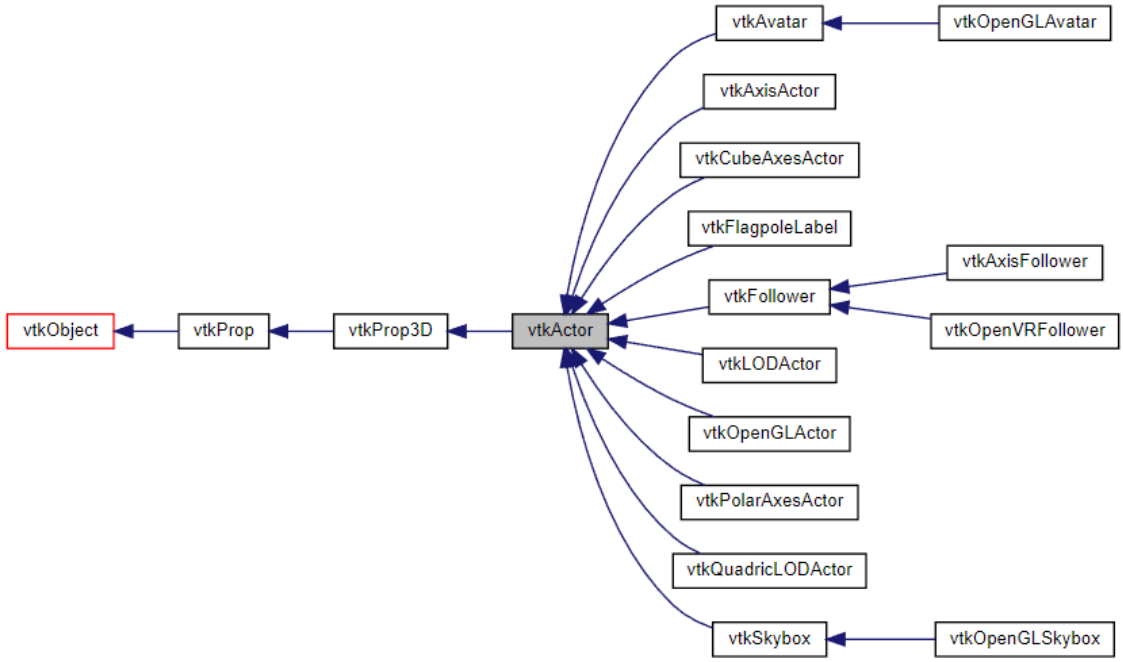

```
############################## ColorModeToMap Scalar##########################

vtkActor actor_bone
    actor_bone SetMapper mapper_bone

vtkActor actor_skin
    actor_skin SetMapper mapper_skin
```

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.

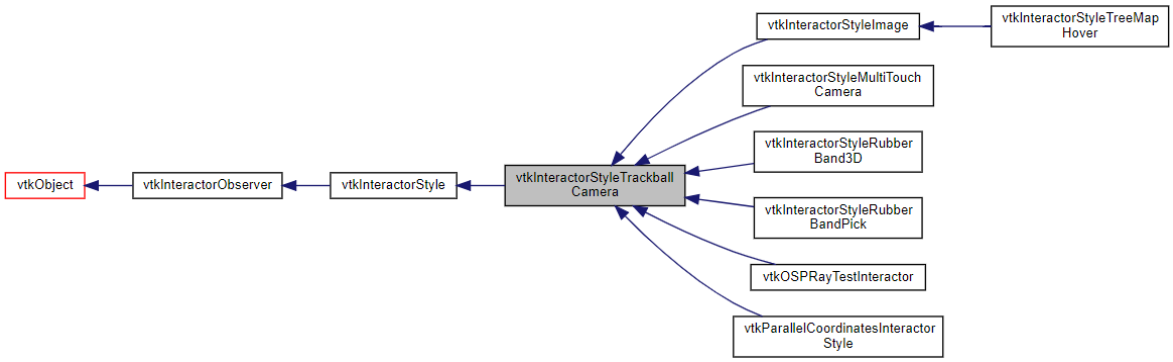
the mapper. This should be a subclass of vtkMapper. Typically vtkPolyDataMapper and vtkDataSetMapper will be used.




```
############################Standard Rendering Stuff ############################

vtkInteractorStyleTrackballCamera style
vtkRenderer ren
vtkRenderWindow renWin
renWin AddRenderer ren
renWin SetSize 800 800

vtkRenderWindowInteractor iren
iren SetRenderWindow renWin
iren SetInteractorStyle style

ren SetBackground 1 1 1
ren AddActor actor_bone
ren AddActor actor_skin
ren ResetCamera
renWin Render

wm withdraw .
```

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.)



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)

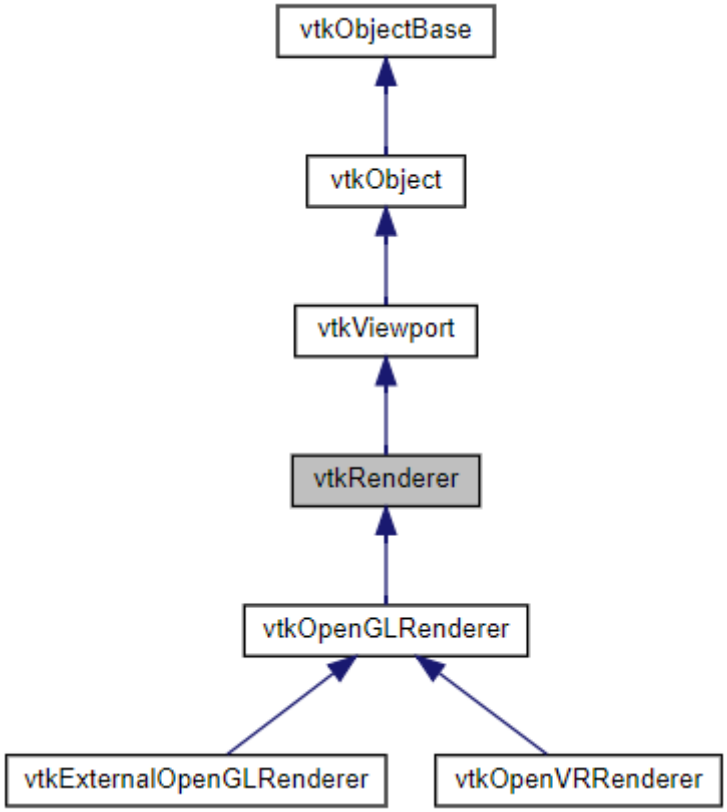
Add/Remove different types of props to the renderer.

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

virtual void vtkRenderer::ResetCamera ( )

Automatically set up the camera based on the visible actors.

The camera will reposition itself to view the center point of the actors, and move along its initial view plane normal (i.e., vector defined from camera position to focal point) so that all of the actors can be seen.



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.

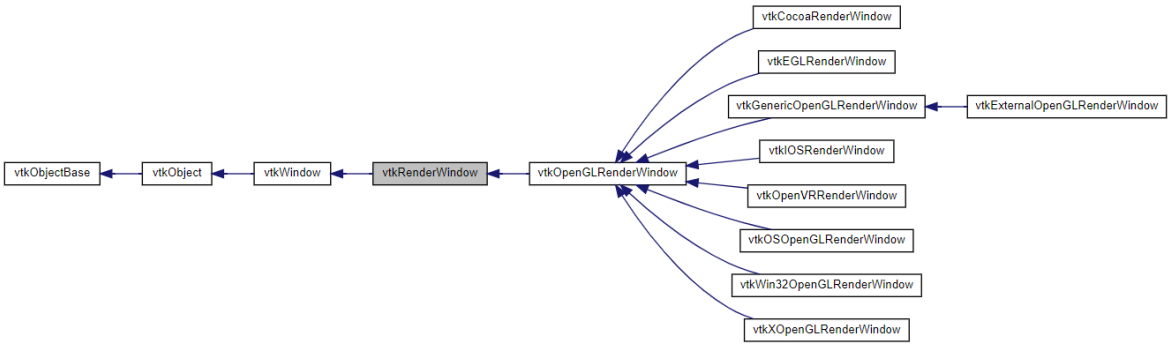
virtual void vtkRenderWindow::AddRenderer (vtkRenderer *)

Add a renderer to the list of renderers.

Reimplemented in **vtkOpenVRRenderWindow**.

void vtkRenderWindow::void vtkRenderWindow::Render ( )Render ( )

Ask each renderer owned by this RenderWindow to render its image and synchronize this process. Implements *vtkWindow*.



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.

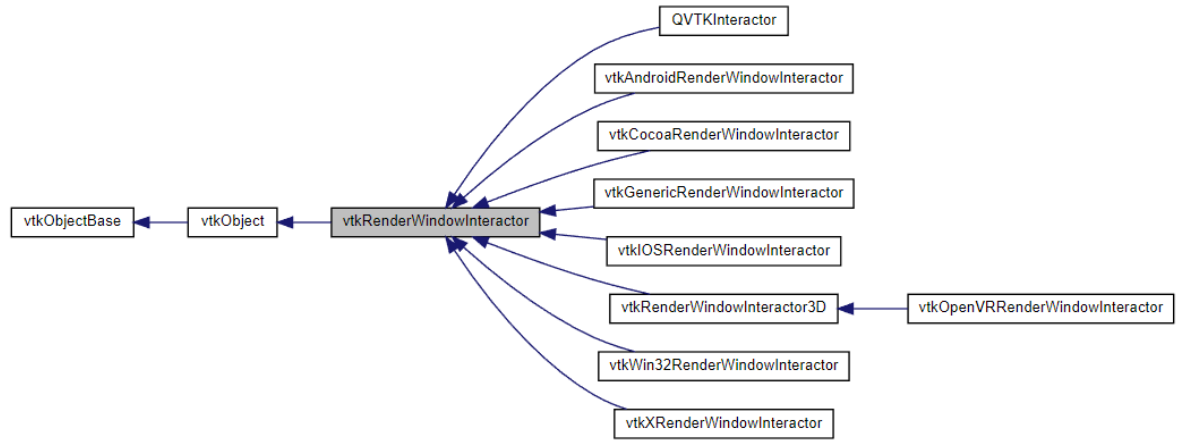
void vtkRenderWindowInteractor::SetRenderWindow (vtkRenderWindow * aren)

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.



Hue, Saturation, and Luminance

Hue

Hue is one of the main properties (called color appearance parameters) of a color, defined technically (in the CIECAM02 model), as "the degree to which a stimulus can be described as similar to or different from stimuli that are described as red, green, blue, and yellow", (which in certain theories of color vision are called unique hues). Hue can typically be represented quantitatively by a single number, often corresponding to an angular position around a central or neutral point or axis on a colorspace coordinate diagram (such as a chromaticity diagram) or color wheel, or by its dominant wavelength or that of its complementary color. The other color appearance parameters are colorfulness, saturation (also known as intensity or chroma), lightness, and brightness.

Usually, colors with the same hue are distinguished with adjectives referring to their lightness or colorfulness, such as with "light blue", "pastel blue", "vivid blue". Exceptions include brown, which is a dark orange.

In painting color theory, a hue is a pure pigment—one without tint or shade (added white or black pigment, respectively). Hues are first processed in the brain in areas in the extended V4 called globs.

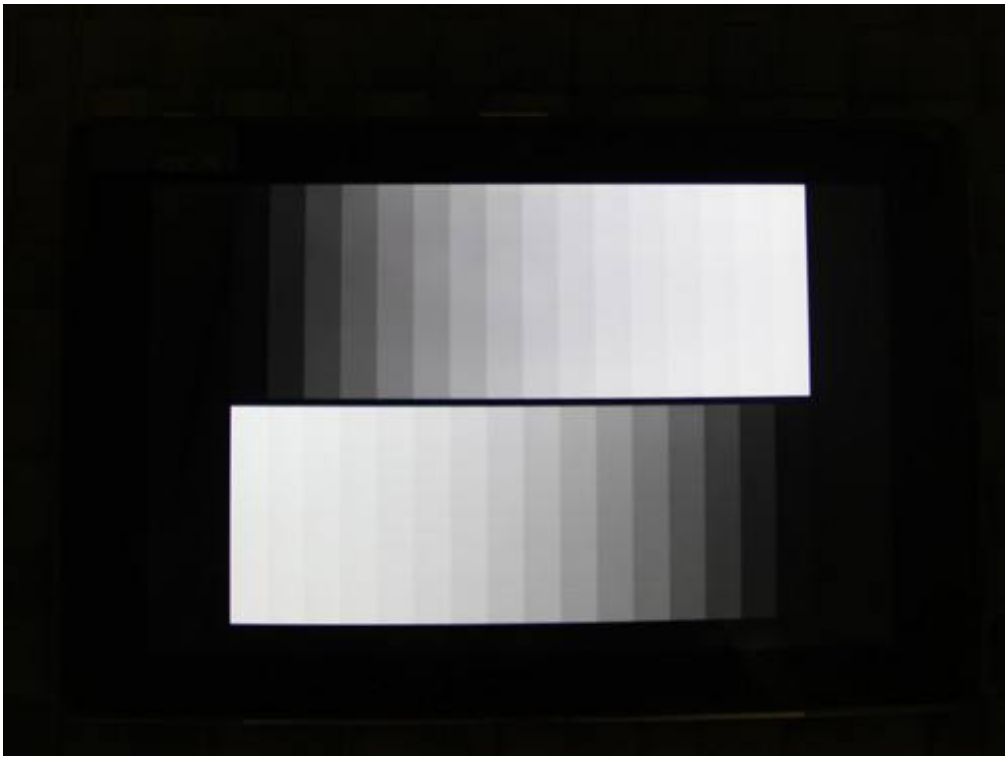
Saturation

The saturation of a color is determined by a combination of light intensity and how much it is distributed across the spectrum of different wavelengths. The purest (most saturated) color is achieved by using just one wavelength at a high intensity, such as in laser light.[citation needed] If the intensity drops, then as a result the saturation drops. To desaturate a color of given intensity in a subtractive system (such as watercolor), one can add white, black, gray, or the hue's complement.

Luminance

Luminance is a photometric measure of the luminous intensity per unit area of light travelling in a given direction. It describes the amount of light that passes through, is emitted from, or is reflected from a particular area, and falls within a given solid angle.

Brightness is the term for the subjective impression of the objective luminance measurement standard (see Objectivity (science) § Objectivity in measurement for the importance of this contrast).

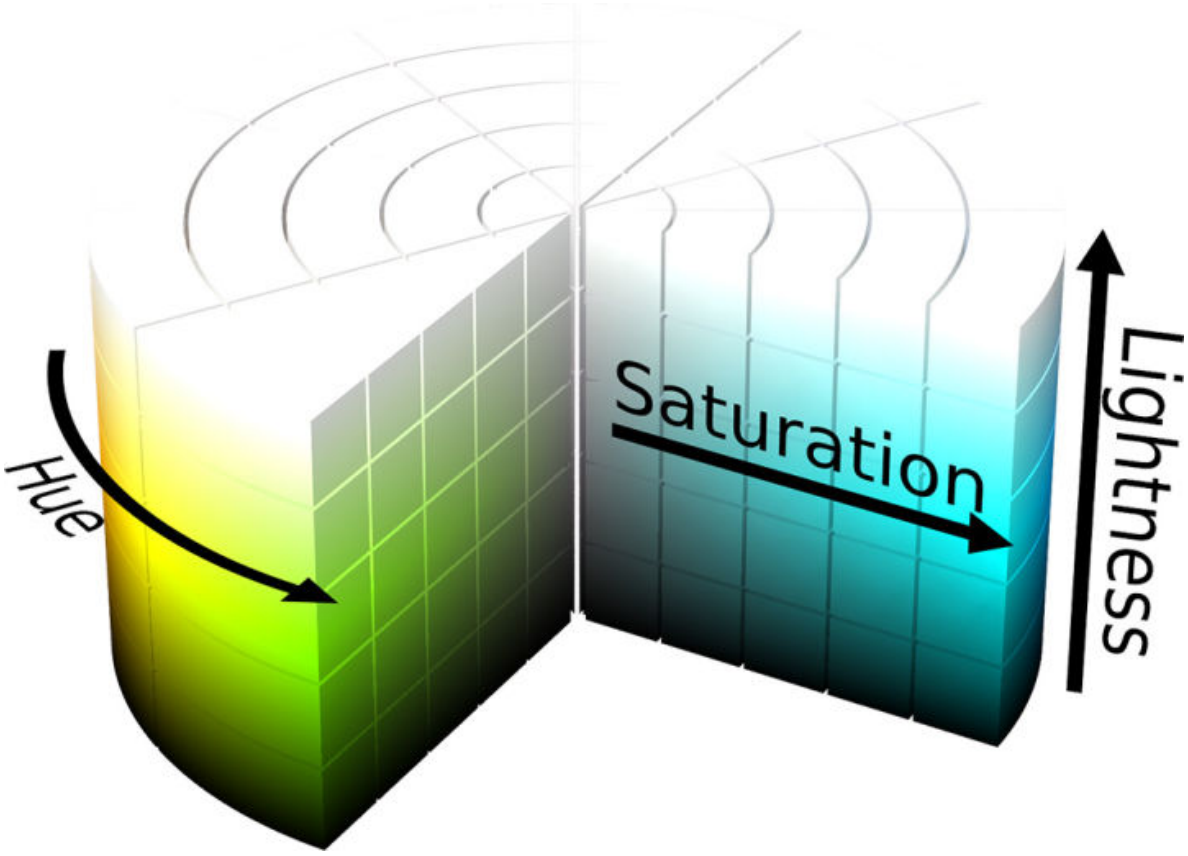


HSL

The H(hue) component of HSL represents the color range perceived by human eyes. These colors are distributed on a plane hue ring, and the value range is 0° to 360° of the central Angle. Each Angle can represent a color. Hue values mean that we can change color by rotating the hue ring without changing the light perception. In practical application, we need to remember the six major colors on the hue ring, as the basic reference: 360°/0° red, 60° yellow, 120° green, 180° blue, 240° blue, 300° magenta, they are arranged in the hue ring according to the interval of 60° central Angle.

The S(saturation) component of HSL refers to the saturation of colors. It describes the changes of color purity under the same hue and brightness with values of 0% to 100%.The larger the value, the less gray in the color, the more vivid the color, showing a change from rational (grayscale) to emotional (pure color).

An HSL L (lightness) component, refers to the brightness of the color, function is to control the brightness of the color change. It also USES the 0% to 100% range. The smaller the value, the darker the color, the closer to black;The larger the number, the brighter the color, the closer to white.



Red, Green, Blue

The RGB color model is an additive color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

To form a color with RGB, three light beams (one red, one green, and one blue) must be superimposed (for example by emission from a black screen or by reflection from a white screen). Each of the three beams is called a component of that color, and each of them can have an arbitrary intensity, from fully off to fully on, in the mixture.

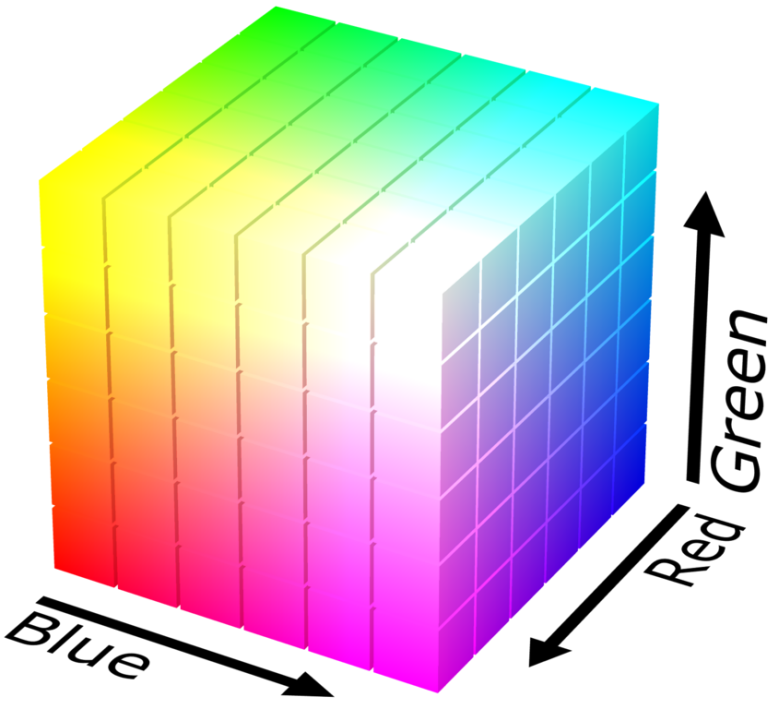
The RGB color model is additive in the sense that the three light beams are added together, and their light spectra add, wavelength for wavelength, to make the final color's spectrum. This is essentially opposite to the subtractive color model, particularly the CMY color model, that applies to paints, inks, dyes, and other substances whose color depends on reflecting the light under which we see them. Because of properties, these three colors create white, this is in stark contrast to physical colors, such as dyes which create black when mixed.

Zero intensity for each component gives the darkest color (no light, considered the black), and full intensity of each gives a white; the quality of this white depends on the nature of the primary light sources, but if they are properly balanced, the result is a neutral white matching the system's white point. When the intensities for all the components are the same, the result is a shade of gray, darker or lighter depending on the intensity. When the intensities are different, the result is a colorized hue, more or less saturated depending on the difference of the strongest and weakest of the intensities of the primary colors employed.

When one of the components has the strongest intensity, the color is a hue near this primary color (red-ish, green-ish, or blue-ish), and when two components have the same strongest intensity, then the color is a hue of a secondary color (a shade of cyan, magenta or yellow). A secondary color is formed by the sum of two primary colors of equal intensity: cyan is green+blue, magenta is blue+red, and yellow is red+green. Every secondary color is the

complement of one primary color; when a primary and its complementary secondary color are added together, the result is white: cyan complements red, magenta complements green, and yellow complements blue.

The RGB color model itself does not define what is meant by red, green and blue colorimetrically, and so the results of mixing them are not specified as absolute, but relative to the primary colors.



HSL and RGB

To convert from HSL or HSV to RGB, we essentially invert the steps listed above (as before, R, G, B $\in$ [0, 1]). First, we compute chroma, by multiplying saturation by the maximum chroma for a given lightness or value. Next, we find the point on one of the bottom three faces of the RGB cube which has the same hue and chroma as our color (and therefore projects onto the same point in the chromaticity plane). Finally, we add equal amounts of R, G, and B to reach the proper lightness.

HSL to RGB

Given a color with hue $H \in [0°, 360°]$, saturation $S_{HSL} \in [0, 1]$, and lightness $L \in [0, 1]$, we first find chroma:

$$C = (1 - |2L - 1|) \times S_{HSL}$$

Then we can find a point (R_1, G_1, B_1) along the bottom three faces of the RGB cube, with the same hue and chroma as our color (using the intermediate value X for the second largest component of this color):

$$H' = \frac{H}{60^\circ}$$
$$X = C \times (1 - |H' \mod 2 - 1|)$$
$$(R_1, G_1, B_1) = \begin{cases} (0, 0, 0) & \text{if } H \text{ is undefined} \\ (C, X, 0) & \text{if } 0 \le H' \le 1 \\ (X, C, 0) & \text{if } 1 \le H' \le 2 \\ (0, C, X) & \text{if } 2 \le H' \le 3 \\ (0, X, C) & \text{if } 3 \le H' \le 4 \\ (X, 0, C) & \text{if } 4 \le H' \le 5 \\ (C, 0, X) & \text{if } 5 \le H' \le 6 \end{cases}$$

Overlap (when H' is an integer) occurs because two ways to calculate the value are equivalent: **X = 0** or **X = C**,as appropriate.

Finally, we can find *R*, *G*, and *B* by adding the same amount to each component, to match lightness:

$$m = L - C/2$$
$$(R, G, B) = (R_1 + m, G_1 + m, B_1 + m)$$

RGB to HSL

This is a reiteration of the previous conversion.

Value must be in range $R, G, B \in [0, 1]$

$$H := \begin{cases} 0, & \text{if } MAX = MIN \Leftrightarrow R = G = B \\ 60^\circ \cdot \left(0 + \frac{G-B}{MAX-MIN}\right), & \text{if } MAX = R \\ 60^\circ \cdot \left(2 + \frac{B-R}{MAX-MIN}\right), & \text{if } MAX = G \\ 60^\circ \cdot \left(4 + \frac{R-G}{MAX-MIN}\right), & \text{if } MAX = B \end{cases}$$

s

$$\text{if } H < 0^\circ \text{ then } H := H + 360^\circ$$

$$S_{\text{HSL}} := \begin{cases} 0, & \text{if } MAX = 0 \Leftrightarrow R = G = B = 0 \\ 0, & \text{if } MIN = 1 \Leftrightarrow R = G = B = 1 \\ \frac{MAX-MIN}{1-|MAX+MIN-1|} = \frac{2MAX-2L}{1-|2L-1|} = \frac{MAX-L}{\min(L,1-L)}, & \text{otherwise} \end{cases}$$

$$L := \frac{MAX + MIN}{2}$$