This notebook is intended to demonstrate how select registration, segmentation, and image mathematical methods of ITKTubeTK can be combined to perform multi-channel brain extraction (aka. skull stripping for patient data containing multiple MRI sequences).

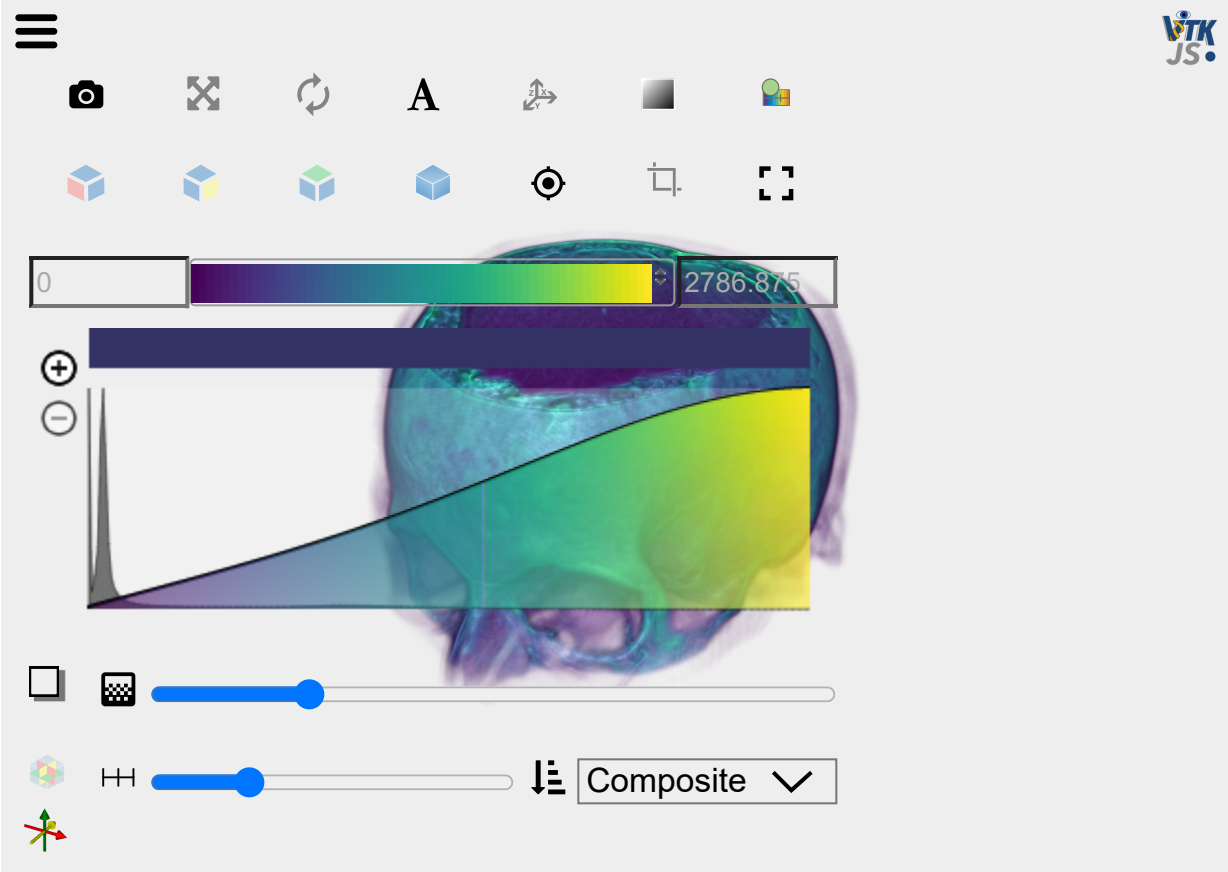
There are many other (probably more effective) brain extraction methods available as open-source software such as BET and BET2 in the FSL package (albeit such methods are only for single channel data). If you need to perform brain extraction for a large collection of scans that do not contain major pathologies, please use one of those packages. This notebook is meant to show off the capabilities of specific ITKTubeTK methods, not to demonstration how to "solve" brain extraction.

In [1]:
```
import itk
from itk import TubeTK as ttk

from itkwidgets import view

import numpy as np
import time
```

In [2]:
```
ImageType = itk.Image[itk.F, 3]

InputBaseName = "../Data/CTP-MinMax/max3"

filename = InputBaseName + ".mha"
im1iso = itk.imread(filename, itk.F)
```

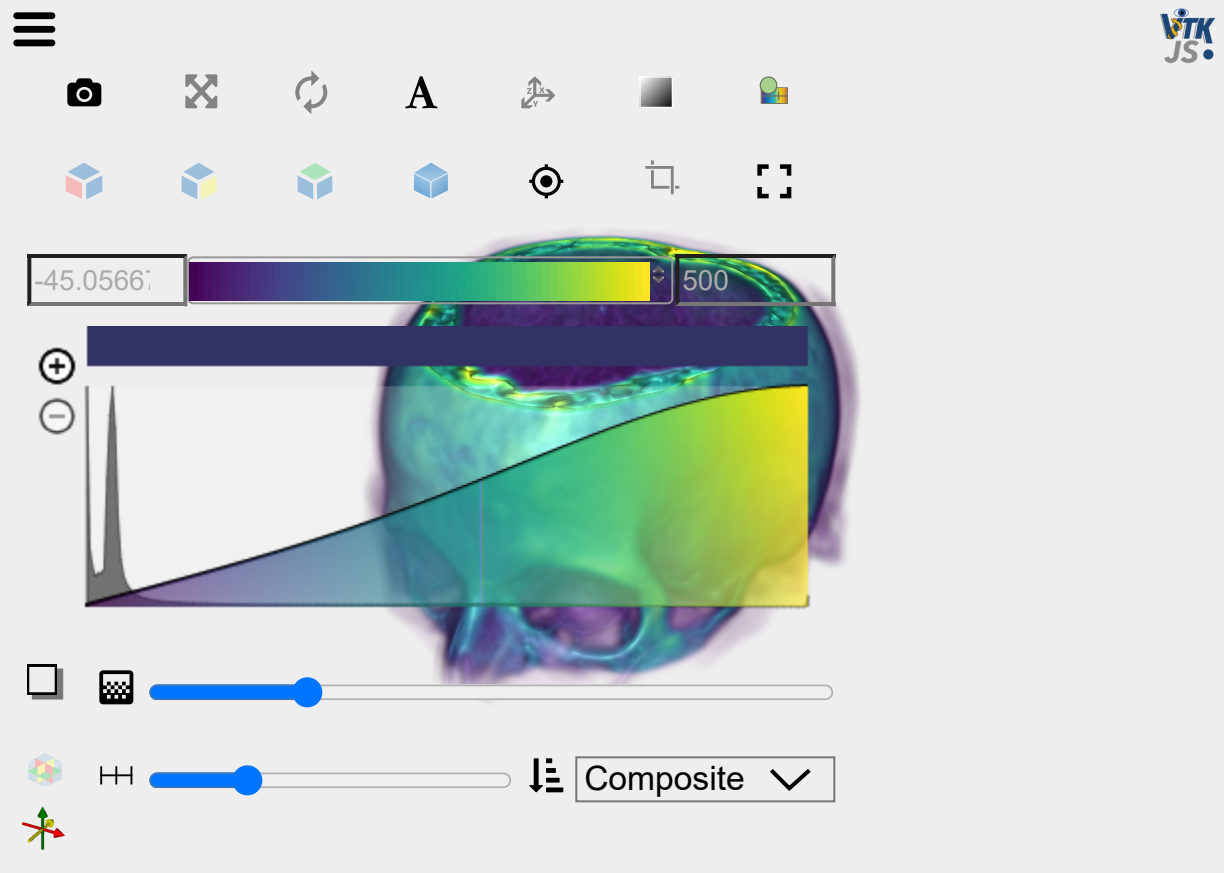
In [3]:
```
N = 8
readerList = ["003", "010", "026", "034", "045", "056", "063", "071"]

imBase = []
imBaseB = []
for i in range(0,N):
    name = "../Data/MRI-Normals/Normal"+readerList[i]+"-FLASH.mha"
    nameB = "../Data/MRI-Normals/Normal"+readerList[i]+"-FLASH-Brain.mha"
    imBaseTmp = itk.imread(name, itk.F)
    imBaseBTmp = itk.imread(nameB, itk.F)
    imBase.append(imBaseTmp)
    imBaseB.append(imBaseBTmp)
```

In [4]:
```
view(im1iso)
```



30

In [5]:
```
imMath = ttk.ImageMath.New(Input=im1iso)
#imMath.Threshold(-4000,-500,1,0)
#headMask = imMath.GetOutput()
imMath.SetInput(im1iso)
#imMath.IntensityWindow(0,1000,1000,0)
#imMath.ReplaceValuesOutsideMaskRange(headMask,-0.5,0.5,-500)
imMath.Blur(1)
imMath.NormalizeMeanStdDev()
imMath.IntensityWindow(-5,5,-500,500)
im1isoBlur = imMath.GetOutput()
view(im1isoBlur)
```

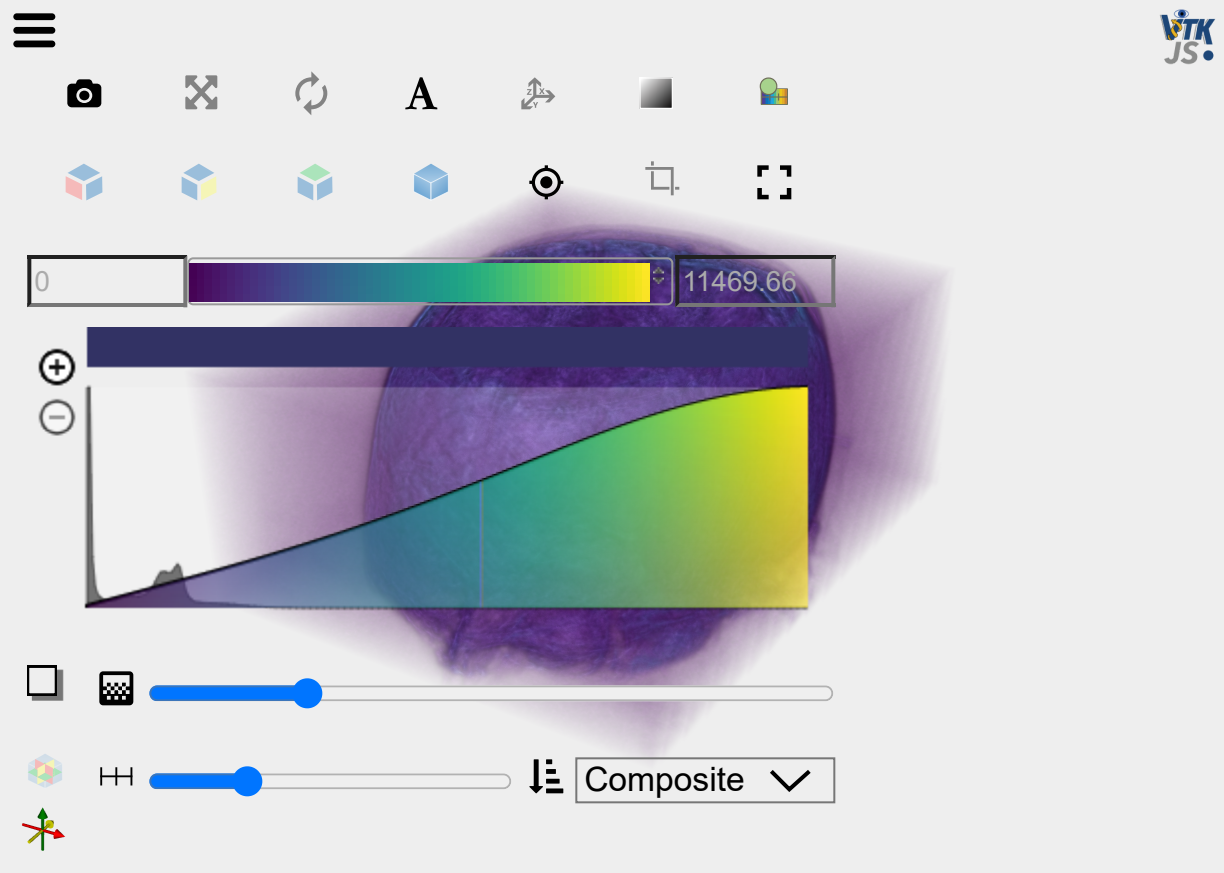


In [6]:
```
RegisterImagesType = ttk.RegisterImages[ImageType]
regB = []
regBB = []
for i in range(0,N):
    start = time.time()

    imMath.SetInput(imBase[i])
    imMath.Blur(1)
    imMath.NormalizeMeanStdDev()
    imMath.IntensityWindow(-5,5,-500,500)
    imBaseBlur = imMath.GetOutput()

    regBTo1 = RegisterImagesType.New(FixedImage=imBaseBlur, MovingImage=im1isoBlu

    regBTo1.SetRigidMaxIterations(3000)
    regBTo1.SetAffineMaxIterations(3000)

    regBTo1.SetExpectedRotationMagnitude(0.2)
    regBTo1.SetExpectedScaleMagnitude(0.25)
    regBTo1.SetExpectedSkewMagnitude(0.01)
    regBTo1.SetExpectedOffsetMagnitude(40)

    regBTo1.SetRigidSamplingRatio(0.1)
    regBTo1.SetAffineSamplingRatio(0.1)

    regBTo1.SetSampleFromOverlap(True)

    regBTo1.SetInitialMethodEnum("INIT_WITH_IMAGE_CENTERS")
    regBTo1.SetRegistration("PIPELINE_AFFINE")
    regBTo1.SetMetric("MATTES_MI_METRIC")

    #regBTo1.SetReportProgress(True)

    regBTo1.Update()

    tfm = regBTo1.GetCurrentMatrixTransform()
    tfmInv = tfm.GetInverseTransform()

    resm = ttk.ResampleImage.New(Input=imBase[i])
    resm.SetMatchImage(im1iso)
    resm.SetTransform(tfmInv)
    resm.SetLoadTransform(True)
    resm.Update()
    img = resm.GetOutput()
    regB.append( img )

    resm = ttk.ResampleImage.New(Input=imBaseB[i])
    resm.SetMatchImage(im1iso)
    resm.SetTransform(tfmInv)
    resm.SetLoadTransform(True)
    resm.Update()
    img = resm.GetOutput()
    regBB.append( img )

    end = time.time()
```

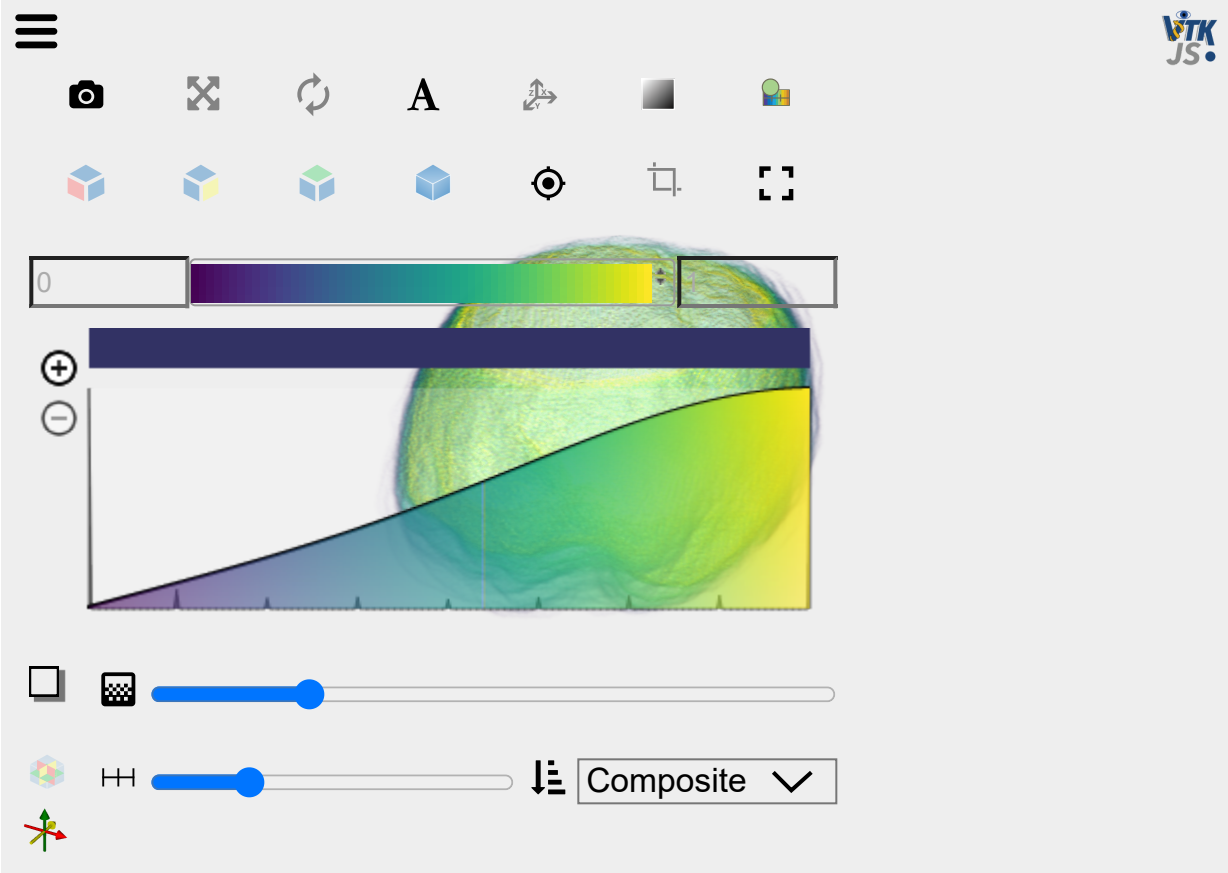
```
    percent = (i + 1) / N * 100
    print('***  ' + str(i) + ' : ' + str(round(percent)) + '% : ' + str(round(enc
```

```
***  0 : 12% : 57s  ***
***  1 : 25% : 57s  ***
***  2 : 38% : 54s  ***
***  3 : 50% : 66s  ***
***  4 : 62% : 67s  ***
***  5 : 75% : 55s  ***
***  6 : 88% : 58s  ***
***  7 : 100% : 68s  ***
```

In [7]:
```
imMath.SetInput(regB[1])
imMath.AddImages(im1iso,20,1)
img = imMath.GetOutput()
view( img )
```

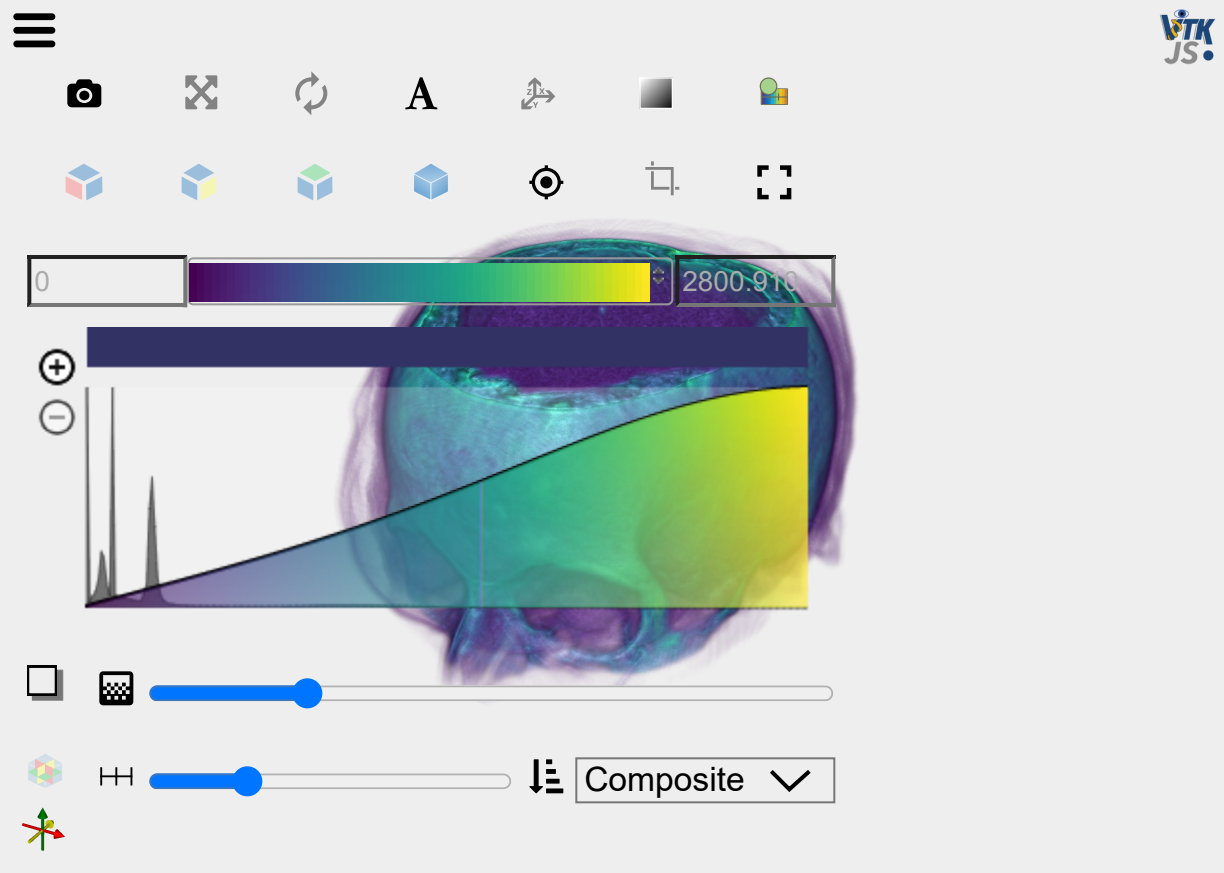


In [8]:
```
regBBT = []
for i in range(0,N):
    imMath.SetInput(regBB[i])
    imMath.Threshold(0,1,0,1)
    img = imMath.GetOutput()
    if i==0:
        imMath.SetInput( img )
        imMath.AddImages( img, 1.0/N, 0 )
        sumBBT = imMath.GetOutput()
    else:
        imMath.SetInput( sumBBT )
        imMath.AddImages( img, 1, 1.0/N )
        sumBBT = imMath.GetOutput()

view(sumBBT)
```



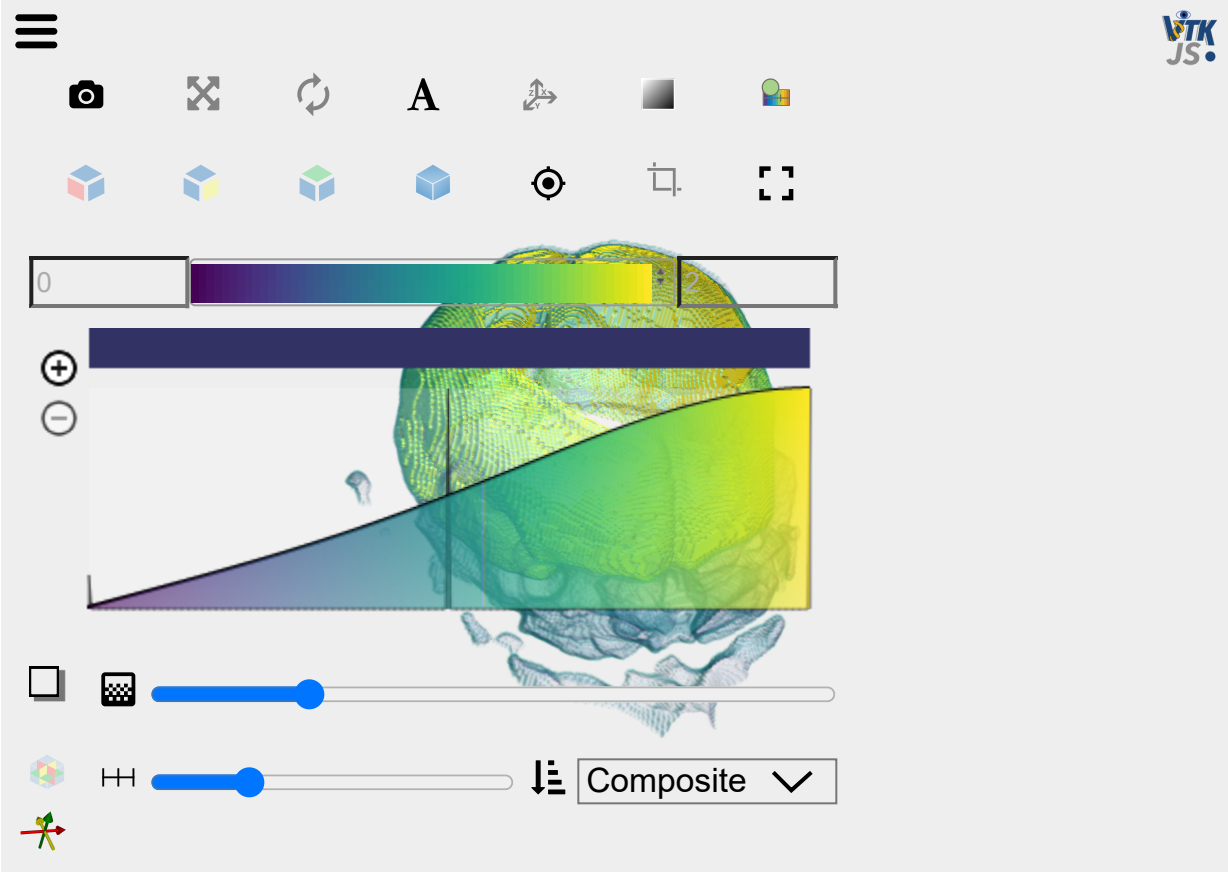
In [9]:
```
imMath.SetInput(sumBBT)
imMath.Threshold(0.85,1.1,1,0)
imMath.Dilate(5,1,0)
imMath.Erode(25,1,0)
brainInside = imMath.GetOutput()

imMath.SetInput( sumBBT )
imMath.Threshold(0,0,1,0)
imMath.Erode(1,1,0)
brainOutsideAll = imMath.GetOutput()
imMath.Erode(20,1,0)
imMath.AddImages(brainOutsideAll, -1, 1)
brainOutside = imMath.GetOutput()

imMath.AddImages(brainInside,1,2)
brainCombinedMask = imMath.GetOutputUChar()
brainCombinedMaskF = imMath.GetOutput()
```

In [10]:
```
imMath.SetInput(brainCombinedMaskF)
imMath.AddImages(im1iso, 100, 1)
brainCombinedMaskView = imMath.GetOutput()
view(brainCombinedMaskView)
```



In [11]:
```
LabelMapType = itk.Image[itk.UC,3]

segmenter = ttk.SegmentConnectedComponentsUsingParzenPDFs[ImageType,LabelMapType]
segmenter.SetFeatureImage( im1iso )
segmenter.SetInputLabelMap( brainCombinedMask )
segmenter.SetObjectId( 2 )
segmenter.AddObjectId( 1 )
segmenter.SetVoidId( 0 )
segmenter.SetErodeDilateRadius( 10 )
segmenter.SetHoleFillIterations( 40 )
segmenter.Update()
segmenter.ClassifyImages()
brainCombinedMaskClassified = segmenter.GetOutputLabelMap()
```

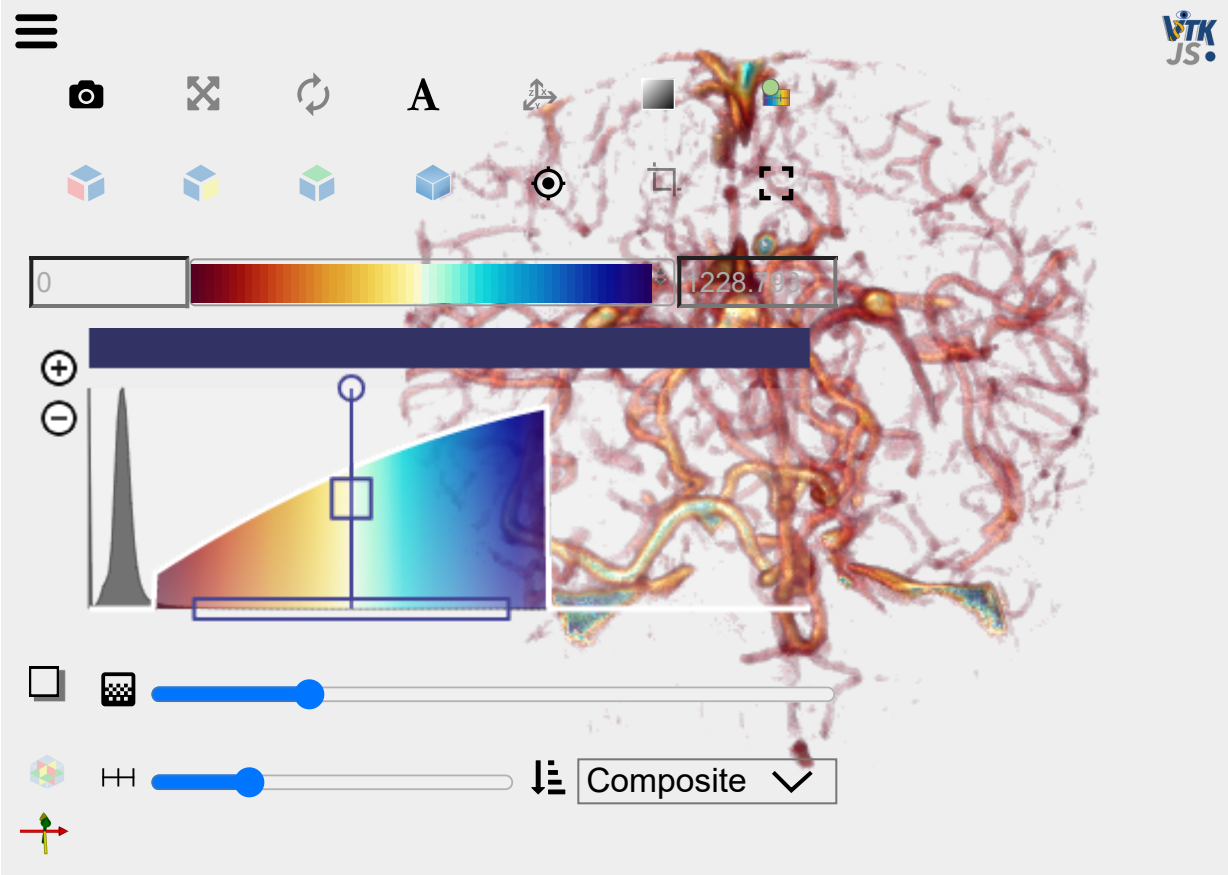
In [12]:
```
view(brainCombinedMaskClassified)
```



30

In [13]:
```
cast = itk.CastImageFilter[LabelMapType, ImageType].New()
cast.SetInput(brainCombinedMaskClassified)
cast.Update()
brainMaskF = cast.GetOutput()

brainMath = ttk.ImageMath[ImageType,ImageType].New(Input = brainMaskF)
brainMath.Threshold(2,2,1,0)
brainMath.Erode(1,1,0)
brainMaskD = brainMath.GetOutput()
brainMath.SetInput( im1iso )
brainMath.ReplaceValuesOutsideMaskRange( brainMaskD, 1, 1, 0)
brain = brainMath.GetOutput()
```

In [14]:
```
view(brain)
```



10

In [15]:
```
writer = itk.ImageFileWriter[ImageType].New(Input = brain)
filename = InputBaseName + "-Brain.mha"
writer.SetFileName(filename)
writer.SetUseCompression(True)
writer.Update()
```

In [ ]: