1. Introduction

1.1 Problem Statement

The objective is to design and develop interactive 3D Visualisation software for reconstructed images. The input is an image stack where each image is a slice of a reconstructed volume along the Z-direction. The function requirement of the software is to produce fully functional UI based interactive volume visualization with features such as clipping, opacity control, slicing etc.

1.2 Use case and Features

- This software is used to visualize 3D computed tomography data. (Tomography is imaging by sections or sectioning, through the use of any kind of penetrating wave.)
- The visualization and quantitative analysis of differently absorbing phases/ surfaces.
- To check for micro porosities, cracks, precipitates or grains in a specimen.
- These visualized objects can be rotated, sliced, clipped and saved as image.
- This software also features a histogram of value populations above the main visualization widget. This is used to show where values were most numerous, and give researchers a general feel for the data. We also have added the opacity function editor. So along with displaying the histogram and the current color map you can interactively edit opacity function, and see the results in the active visualization window.
- This software also provides an option of applying different color maps. One can also change the color resolution of the color map for a particular range of gray values.
- Slices of the object can be obtained using the plane widget which can be rotated, moved and scale in any angle and direction. The same plane can be used to clip the object too.
- Clipping can also be done using a clipping box.

1.3 Input Data

Reconstructed image stack obtained by computed tomography is the input for this software.

The reconstruction of an image from the acquired data is an inverse problem. Often, it is not possible to exactly solve the inverse problem directly. In this case, a direct algorithm has to approximate the solution, which might cause visible reconstruction artefacts in the image. Iterative algorithms approach the correct solution using multiple iteration steps, which allows to obtain a better reconstruction at the cost of a higher computation time.

1.4 Volume Rendering

Volume rendering is a set of techniques used to display a 2D projection of a 3D discretely sampled data set, typically a 3D scalar field. A typical 3D data set is a group of 2D slice images acquired by a for example CT, MRI, or MicroCT scanner. Usually these are acquired in a regular pattern (e.g., one slice every millimetre) and usually have a regular number of image pixels in a regular pattern.

This is an example of a regular volumetric grid, with each volume element, or voxel represented by a single value that is obtained by sampling the immediate area surrounding the voxel.

To render a 2D projection of the 3D data set, one first needs to define a camera in space relative to the volume. Also, one needs to define the opacity and color of every voxel. This is usually defined using an RGBA (for red, green, blue, alpha) transfer function that defines the RGBA value for every possible voxel value.

For example, a volume may be viewed by extracting isosurfaces (surfaces of equal values) from the volume and rendering them as polygonal meshes or by rendering the volume directly as a block of data. The marching cubes algorithm is a common technique for extracting an isosurface from volume data. Direct volume rendering is a computationally intensive task that may be performed in several ways.

2. Framework and Libraries

Programming language used for coding this software is python. It is because of the simplicity of python, availability of good libraries for achieving your task, ease of using them, good community of users using them.

Python version: **Python 2.7.12**

2.1 VTK

VTK version: **5.10.1**

The Visualization Toolkit (VTK) is an open-source, freely available software system for 3D computer graphics, image processing, and visualization. It consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java, and Python. VTK supports a wide variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods, as well as advanced modelling techniques such as implicit modelling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation.

VTK has an extensive information visualization framework and a suite of 3D interaction widgets. The toolkit supports parallel processing and integrates with various databases on GUI toolkits such as Qt and Tk.

VTK is cross-platform and runs on Linux, Windows, Mac, and Unix platforms. VTK is part of Kitware's collection of commercially supported open-source platforms for software development.

VTK applications are largely constructed by connecting vtkAlgorithms together. Each algorithm (or filter) inspects the dataset or datasets it is given and produces some derived data for the algorithm (or algorithms) connected to it. The connected set of filters forms a data-flow network. VTK uses reference counting heavily to eliminate redundant memory consumption and timestamps in the demand-driven network to eliminate redundant computation. Algorithms are strongly type-checked to enforce compatible filter connectivity.

2.2 PyQt

Qt version: **4.8.7**

PyQt is widely used for developing graphical interfaces that can be run on various operating systems. It's one of the most popular GUI choices for Python programming.

PyQt brings together the Qt C++ cross-platform application framework and the cross-platform interpreted language Python.

Qt also includes Qt Designer, a graphical user interface designer. PyQt is able to generate Python code from Qt Designer. It is also possible to add new GUI controls written in Python to Qt Designer.

Python is a simple but powerful object-orientated language. Its simplicity makes it easy to learn, but its power means that large and complex applications can be created. Its interpreted nature means that Python programmers are very productive because there is no edit/compile/link/run development cycle.

PyQt combines all the advantages of Qt and Python. A programmer has all the power of Qt, but is able to exploit it with the simplicity of Python.

2.3 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.

Matplotlib has many predefined color maps which are used in this software.

Also for plotting the histogram and the opacity function matplotlib's functions are used.

Matplotlib can be used to create histograms. A histogram shows the frequency on the vertical axis and the horizontal axis is another dimension. Usually it has bins, where every bin has a minimum and maximum value. Each bin also has a frequency between x and infinite.

3. The Visualization Toolkit

3.1 Basic objects

In the Visualization Toolkit there are seven basic objects that we use to render a scene:

1. **vtkRenderWindow**: manages a window on the display device, one or more renderers draw into an instance of vtkRenderWindow .

2. **vtkRenderer**: coordinates the rendering process involving lights, cameras, and actors.

3. **vtkLight**: a source of light to illuminate the scene.

4. **vtkCamera**: defines the view position, focal point, and other viewing properties of the scene.

5. **vtkActor** : represents an object rendered in the scene, including its properties and position in the world coordinate system. (Note: vtkActor is a subclass of vtkProp. vtkProp is a more general form of actor that includes annotation and 2D drawing classes.)

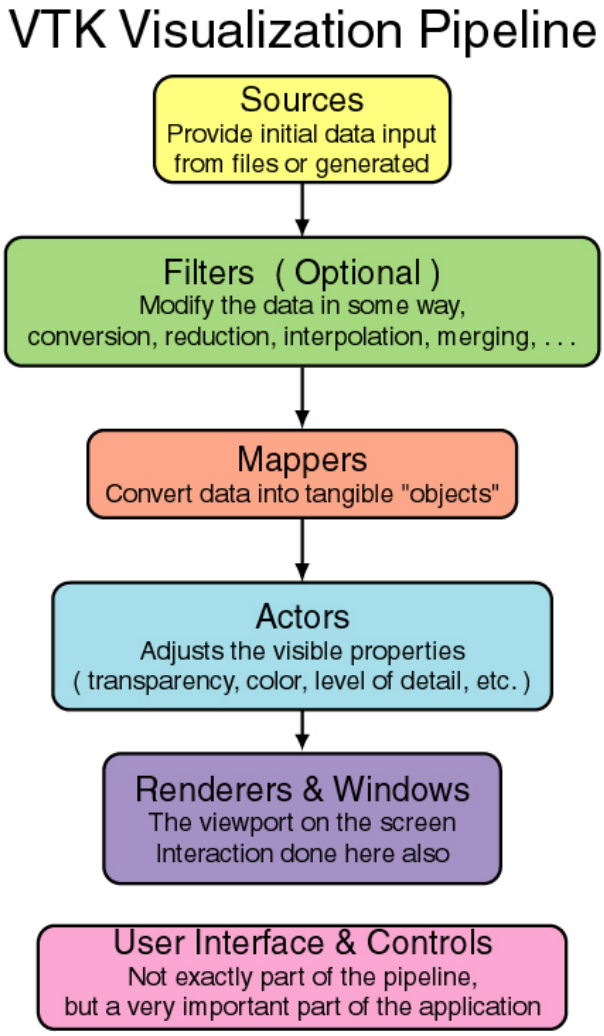
6. **vtkProperty** : defines the appearance properties of an actor including color, transparency, and lighting properties such as specular and diffuse. Also representational properties like wireframe and solid surface.

7. **vtkMapper** : the geometric representation for an actor. More than one actor may refer to the same mapper.

3.2 The Visualization Pipeline

Visualization transforms data into images that efficiently and accurately convey information about the data. Thus, visualization addresses the issues of transformation and representation.

Transformation is the process of converting data from its original form into graphics primitives, and eventually into computer images. This is our working definition of the visualization pro-cess. An example of such a transformation is the process of extracting stock prices and creating an x-y plot depicting stock price as a function of time.



3.3 Types of Datasets

Five datasets are implemented in VTK: vtkPolyData , vtkImageData , vtkStructuredGrid , vtkRectilinearGrid , and vtkUnstructuredGrid

1. Polygonal Data

These primitives also are frequently generated or consumed by computational geometry and visualization algorithms. In the Visualization Toolkit , we call this collection of graphics primitives polygonal data . The polygonal dataset consists of vertices, polyvertices, lines, polylines, polygons, and triangle strips. The topology and geometry of polygonal data is unstructured, and the cells that compose that dataset vary in topological dimension. The polygonal dataset forms a bridge between data, algorithms, and high-speed computer graphics.

2. Image Data

An image dataset is a collection of points and cells arranged on a regular, rectangular lattice. The rows, columns, and planes of the lattice are parallel to the global x-y-z coordinate system. If the points and cells are arranged on a plane (i.e., two-dimensional) the dataset is referred to as a pixmap, bitmap, or image. If the points and cells are arranged as stacked planes (i.e., three-dimensional) the dataset is referred to as a volume. Keep in mind that the term image data refers to

images, volumes, or one-dimensional point arrays collectively. Structured points was the terminology used in earlier versions of VTK.

3. Rectilinear Grid

The rectilinear grid dataset is a collection of points and cells arranged on a regular lattice. The rows, columns, and planes of the lattice are parallel to the global x-y-z coordinate system. While the topology of the dataset is regular, the geometry is only partially regular. That is, the points are aligned along the coordinate axis, but the spacing between points may vary.

4. Structured Grid

A structured grid is a dataset with regular topology and irregular geometry. The grid may be warped into any configuration in which the cells do not overlap or self-intersect. The topology of the structured grid is represented implicitly by specifying a 3-vector geometry dimensions . The n x n y n z is explicitly represented by maintaining an array of point coordinates. The composing cells of a structured grid are quadrilaterals (2D) or hexahedron (3D). Like image data, the structured grid has a natural coordinate system that allows us to refer to a particular point or cell using topological i-j-k coordinates.

5. Unstructured Grid

The most general form of dataset is the unstructured grid. Both the topology and geometry are completely unstructured. Any cell type can be combined in arbitrary combinations in an unstructured grid. Hence the topology of the cells ranges from 0D (vertex, polyvertex) to 3D (tetrahedron, hexahedron, voxel). In the Visualization Toolkit any dataset type can be expressed as an unstructured grid. We typically use unstructured grids to represent data only when absolutely necessary, because this dataset type requires the most memory and computational resources to represent and operate on.

4. Code Documentation

Classes:

4.1 Driver.py

This class manages calls to different class methods (/ utilities) as and when required and serves as a central module. This class should be called in order to run the entire software. It takes the UI from MainWindowUi class.

The external libraries and classes that it imports are:

- Project Classes : LoadStack, MainWindowUi, Renderer, CutObj, HistogramWidget
- QtCore,QtGui from PyQt4
- matplotlib
- numpy
- vtk_to_numpy from vtk.util.numpy_support
- sys, os, time
- threading

Methods:

__init__(self)

It initializes and setups the UI and connects all the buttons to their respective callback functions for the action event. And sets the initial state of the buttons (some are disabled).

openLoadStackDialog(self)

It calls the function showDialog() from LoadStack class. It displays the dialog box to take the information of input image stack. After the user press ‘OK’, an internal call is made to the renderVolume() function of Renderer class which renders the output object.

renderV(self)

It calls the renderVolume() of Renderer class. Changes the states of buttons (enables/disables) and also gives internal call to function that plots histogram and alpha function.

cutObj(self)

Adds the plane of plane widget as a clipping plane to the volume by calling cut() function of CutObj class.

toggleOutline(self)

Changes the visibility of the outline actor.

togglePlane(self)

On / off the plane widget. Also accordingly changes the visibility of slice. And enables/ disables the option of setting the slice resolution.

toggleClippingBox(self)

On / off the box widget. This is used for clipping the volume.

toggleVolume(self)

Toggles the visibility of rendered volume.

toggleSlice(self)

Toggles the visibility of the contour actor, which displays the slice of the volume captured by plane widget.

toggleAlignedPlane(self)

On / Off the ImagePlaneWidget widget. By default aligns the plane to Z axis and sets the state of buttons and slider accordingly.

alignPlaneToX(self)

Aligns the ImagePlaneWidget to X axis. Sets range of the slider according to the values of data extent taken from the reader. Sets the default value of slider to the center index of the data extent and places the plane at the center of the object in X axis.

alignPlaneToY(self)

Aligns the ImagePlaneWidget to Y axis. Sets range of the slider according to the values of data extent taken from the reader. Sets the default value of slider to the center index of the data extent and places the plane at the center of the object in Y axis.

alignPlaneToZ(self)

Aligns the ImagePlaneWidget to Z axis. Sets range of the slider according to the values of data extent taken from the reader. Sets the default value of slider to the center index of the data extent and places the plane at the center of the object in Z axis.

valueChangeSlider(self)

Takes the new value of the slider set by the user and sets it as the new position of the plane in the object. It is called when slider is moved.

changeColorMap(self)

Gives call to HistogramWidget class's showDialogBox() function which displays a dialog box asking the new color map to choose and set the color resolution if needed.

takeSnapshot(self)

Asks for the path to select for saving the snapshot and calls the snapshot() from Renderer class.

plot(self)

Adds a subplot to the figure of histogram widget and calls the plotHist() and plotAlphaFunc() from HistogramWidget class to plot the histogram of image stack.

setSliceResolution(self)

Sets the resolution of the slice of the volume obtained from the plane widget.

addPoint(self)

Adds a point to the alpha function and also to the alpha function plotted over the histogram.

removePoint(self)

Removes a point from the alpha function and also from the alpha function plotted over the histogram.

createTable(self)

Creates a table from the values present in the dictionary alphaFuncPoints. Creates it in sorted order.

changeBGColor(self)

Displays a color picker and changes background color of render window to the color selected by the user.

record(self)

Records the interaction with the object in form of image sequence. It actually takes multiple snapshots at the given frame rate. It creates a new thread for it because the MainWindow should remain active in order to do interaction with rendered object.

task1(self, fname, nImg, sTime)

This is the task carried out by the thread. It clicks multiple snapshot (takes nImg images, each image after sTime(sleep time))(calculated from frame rate and time of recording taken from text boxes) and saves it with file name prefix fname.

resetCamera(self)

This function aligns the camera at particular location.

Some important variables:

flag_stackRendered

This keeps a check on whether the stack is rendered or not. Initially set to False, turns True only when volume is rendered at least once. This helps in deciding whether or not to make a call to a particular function that makes some changes in the volume.

alphaFuncPoints

This is a dictionary which stores the alpha function points that is in form of gray value as key and the corresponding opacity as value. This helps in maintaining the table in a sorted way.

cMin and **cMax**

This stores the min and max of the range of color map resolution set by the user (variables with same name exist in Renderer class, those are the original values of minimum and maximum gray values, combination of these two is used in order to plot the alpha function).

4.2 Renderer.py

This class will take the information of the image stack(such as it's path, number of images,etc.) from the LoadStack object or the vtkObject (pickle) and pass it to the VTK pipeline. Volume's default properties and functions(color and alpha) are defined here. Volume will be mapped, rendered and the output will be given to the render window of vtkWidget. All the other widgets like plane widget for clipping and slicing, box widget for clipping box and snapshot are defined here.

It imports :

- vtk
- HistogramWidget

Methods :

readImgStack(self, loadedDataInfo)

This function fetches the information of the image stack from LoadStack object and reads it using the TiffReader of vtk. It also sets the cMin and cMax depending on whether the image is 8 bit or 16 bit.

renderVolume(self, vtkWidget)

This function is like heart of whole software. It actually handles the entire VTK pipeline. Creates color function and alpha function. Sets it to the volume mapper and also takes the output from reader as input to volume mapper. Creates the volume. Creates outline actor, plane widget, slice, box widget and the axes actor. Handles the interaction events of plane widget and box widget for slicing and clipping purpose. Gives all this output to vtkWidget of the MainWindowUi.

renderBlank(self,vtkWidget)

When the driver program loads, this function is called, which just sets the background and renders no object

snapshot(self,fname)

This function is used to take the snapshot of whatever is in the camera view of render window and saves it with the fname

4.3 HistogramWidget.py

This class takes care of the histogram widget. It provides function to apply color maps to the rendered volume and also the histogram. It also plots the alpha function over the histogram.

It imports:

- matplotlib.pyplot, matplotlib.colors
- QtGui,QtCorefrom PyQt4
- numpy
- Ui_Dialog from ColorMapDialogUi
- sys

Methods:

showDialogBox(self, driver)

It initializes the dialog with the UI designed in ColourMapDialogUi which asks user to chose color map from the combo-box and enter the color resolution if required.

accept(self)

This function is called when the 'OK' button of dialog box is pressed. It sets the new cMin and cMax of the Driver class depending on the color resolution range entered by user. It sets the new color map to the color transfer function of Renderer class and also plots a new histogram with the new color map.

plotHist(self, driver, cName, cMin, cMax)

It plots a histogram of all the gray values (taken from the vtkTIFFReader outpput, which is converted into numpy array) using the matplotlib functions. Also color for every patch is set using the color map.

plotAlphaFunc(self, func, ax, canvas, cMin, cMax)

It plots the opacity function over the histogram using the twin axis and the interpolated values directly from the piecewise function.

setColorMap(self, cFunc, cName, cMin, cMax)

Sets the color map to the color transfer function of Renderer class according to the color map name and color resolution range (cMin - cMax).

4.4 LoadStack.py

This class takes the input from the user ,i.e, details about the image stack to loaded. It provides all this info to the Renderer. It imports:

- QtGui,QtCore from PyQt4
- vtk
- sys, os
- Ui_LoadDialog from LoadStackDialogUi

Methods:

showDialog(self)

It initializes the dialog box with the UI designed in LoadStackDialogUi. Buttons are connected to their corresponding callback functions for the action events.

setDirPath(self)

Opens a File Dialog box to browse and locate the path of directory from where the input image stack will be loaded.

accept(self)

This method is called when "Ok" is clicked. It assigns the values entered by the user in the text box to the class variables. It also calls the renderV() of Driver class in order to start the rendering of the input image stack.

getExtent(self)

This method reads the first image and returns a tuple having info about the extents of the image.

getDigits(self)

It return the length of this offset as the number of digits in file name. This helps the SetFilePattern() in the PRenderer to read the files having specific digit pattern (/ format specifier) in the file name.

4.5 CutObj.py

This class will cut (/actually clip by adding a clipping plane) the rendered object based on the plane position of PlaneWidget. Calling the plane widget is necessary before calling the cut() method of this class.(which has been taken care of in the logic of Driver class)

Methods:

cut(self,pRenderer)

Adds current plane of the plane widget as clipping plane to the volume mapper.

4.6 MainWindowUi.py

This UI class setups the UI for the Driver as the Main Window using setupUi(self, MainWindow) method. It has a Grid Layout. Contains Histogram Widget, table to modify the alpha function, vtk widget and a frame which has many options for interacting with the rendered object.

It imports:

- QtCore, QtGui from PyQt4
- QVTKRenderWindowInteractor from vtk.qt4.QVTKRenderWindowInteractor
- FigureCanvasQTAgg, NavigationToolbar2QT from matplotlib.backends.backend_qt4agg
- Figure from matplotlib.figure

4.7 LoadStackDialogUi.py

This UI class setups UI for the dialog box which asks for the details about the input image stack. Information like the stack's directory, file name prefix, offset value, number of images to load and the data spacing is asked from the user.

It imports QtCore, QtGui from PyQt4

4.8 ColorMapDialogUi

UI design for the dialog box which asks for the new color map to select from the combo-box and also if the user want to change the color resolution range, it provides text boxes to enter the range.

It imports QtCore, QtGui from PyQt4

5. Mapping of UI items

5.1 MainWindowUi

Name	Text	Callback function
pushButton	Load Image Stack	openLoadStackDialog
pushButton_2	Clip (*using plane widget)	cutObj
pushButton_3	Set	setSliceResolution
pushButton_4	Change Color Map	changeColorMap
pushButton_5	Snapshot	takeSnapshot
pushButton_7	Record	record
pushButton_8	Remove	removePoint
pushButton_9	Add	addPoint
pushButton_10	Reset Volume	renderV
pushButton_11	X	alignPlaneToX
pushButton_12	Change BG Color	changeBGColor
pushButton_13	Reset Camera	resetCamera
pushButton_14	Y	alignPlaneToY
pushButton_15	Z	alignPlaneToY
radioButton	Outline	toggleOutline
radioButton_2	Plane Widget	togglePlane
radioButton_3	Clipping Box	toggleClippingBox
radioButton_4	Volume Visibility	toggleVolume
radioButton_5	Slice Visibility	toggleSlice
radioButton_6	Aligned Plane Widget	toggleAlignedPlane
lineEdit	300	*For slice resolution
lineEdit_2		*For intensity value of alpha function
lineEdit_3		*For opacityvalue of alpha function
lineEdit_4	10	* Time of recording
lineEdit_5	16	*Frame rate
horizontalSlider	*for positioning image plane widget	valueChangeSlider

5.2 LoadStackDialogUi

Name	Text	Callback function
buttonBox	Contains two standard buttons:	
	Ok button	accept (*overridden method)
	Cancel button	reject (*of parent Dialog class)
pushButton	Browse	setDirPath
lineEdit		* directory path
lineEdit_2		* file prefix
lineEdit_3		* offset
lineEdit_4		* Number of images
lineEdit_5		* Data spacing for x
lineEdit_6		* Data spacing for y
lineEdit_7		* Data spacing for z

5.3 ColorMapDialogUi

Name	Text	Callback function
buttonBox	Contains two standard buttons:	
	Ok button	accept (*overridden method)
	Cancel button	reject (*of parent Dialog class)
radioButton	Set color resolution range :	enableSettingRange
lineEdit	0 (default)	* cMin (for color resolution range)
lineEdit_2	65535	* cMax
comboBox	* names of all color maps	

6. Problems faced and Future Scope

- **Getting the Data extent without asking to user**

  In the LoadStack class created a function that reads the first file, gets the x and y data extent and number of images is the z extent.

- **Data Spacing**

  Not sure about what values to pass as x, y and z data spacing. So for now its upto user to set spacing at the time of loading the input.

- **Image name having specific format of digit pattern**

  There was a problem if the image stack have a particular number of digits in file name (like some may have 3 digits Img_000, Img_001.. or some may be normal like Img_0,Img_1,..) now its a normal human tendency that one will follow this pattern while giving the offset (means if image name has the digit pattern as Img_000,Img_001 so one will write the offset as "000"). Assuming this we return the length of this offset as the number of digits in file name. This helps the SetFilePattern() in the PRenderer to read the files having such specific digit pattern in the file name.

- **Unable to use same plane for Clipping and Slicing**

  For clipping the vtkMapper (AddClippingPlane()) takes the input object of vtkPlane class (obtained from planeWidget.GetPlane()).

  To display the slice we use a vtkActor which takes the input as a vtkPolyData (which can not be obtained from vtkPlane), so for now two planes are created, one of vtkPlane (used for clipping) and other vtkPolyData (which takes input from planeWidget.GetPolyData() which is then passed to the vtkActor) used for slicing.

- **Segmentation fault whenever ‘Reset Volume’ was pressed.**

  If we are interacting with the volume using any widget (plane, clipping box, image plane), if we press the ‘Reset Volume’ button with any of those widget ON it gave Segmentation fault.

  So created a flag ‘flag_stackRendererd’ to have check whether the stack is rendered at least once. If rendered, then if turns OFF all the widgets first and then the call to render volume to default is given.

- **Maintaining sorted order of alpha function point in Qtable**

  Qtable don’t allow to store vales in form of key-value pair. If given sort function it sorts only one column and not the corresponding row.

  So created a dictionary, which maintains the alpha value function points. This is then used to display the points in correct order in the table and also removes duplicity issue.

- **Change in alpha function plot after changing the color map resolution range**

  Used two variables cMin and cMax. One being variables of Renderer class that store original range of grey values. Other, variables of Driver class which changes according to the range entered by the user. Then all the values other than in the range are made zero in alpha function.

- **Main window turns inactive while recording**

  While taking the multiple snapshot the main window was turning inactive and interaction with the object was not possible. So, created a different thread and assigned it the task of taking snapshots. This made the main window to remain active as the loop for taking snapshot was running in different thread.

- **Unable to display status of recording**

  The main window GUI running in main thread doesn't allow any other GUI to be created in other thread. It is giving an error "QPixmap: It is not safe to use pixmaps outside the GUI thread". Tried this for QProgressBar and also for QDialogBox/MessageBox but didn't worked.

- **X-Y-Z plot in different render window**

  Wanted to display the different projection of the object in the 3 planes (X-Y, Y-Z, X-Z). But, unable to get the surface poly data of the plane widget in 2D form, the slice displayed in the different render window was actually the projection of that slice with respect to the camera view.

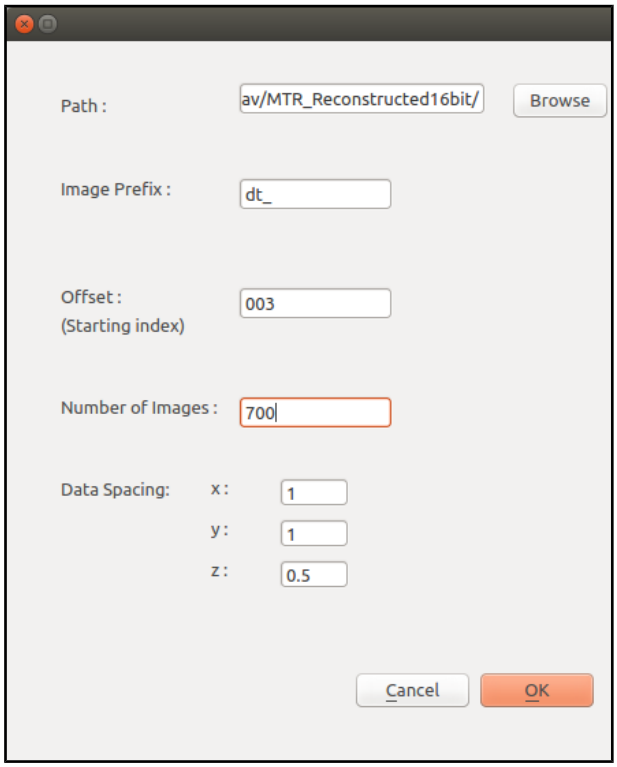
- **Interactive histogram**

  Make the histogram plot more interactive. One should be able to add the point to the alpha function by just clicking on the line. User should able to interact with those points too, like moving them, etc.

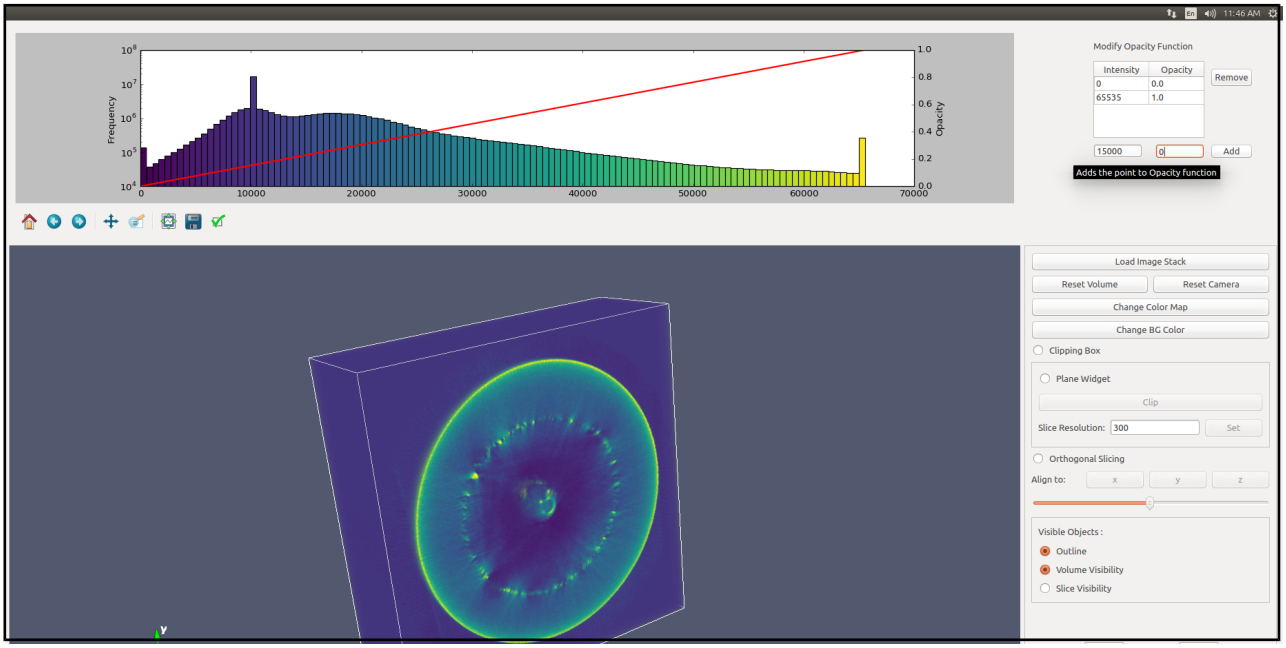
7. UI Screenshots and functionalities

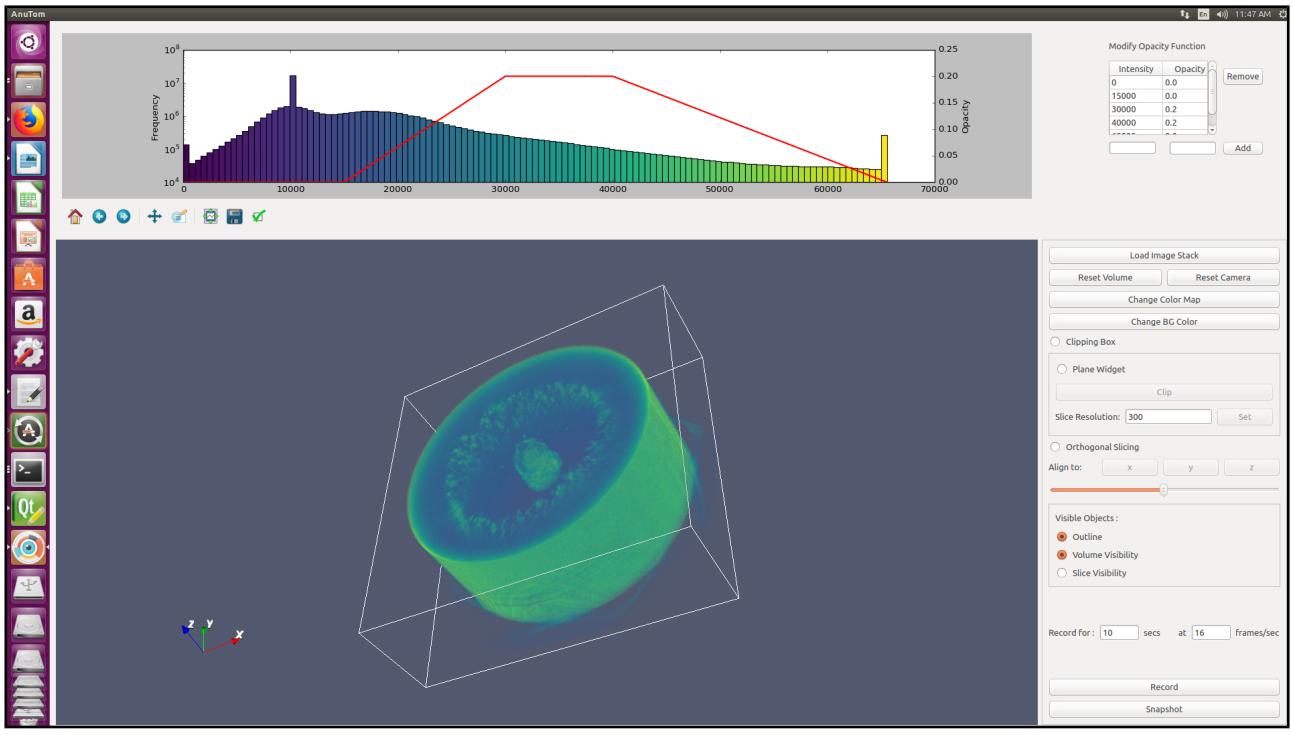
- **Initial Window**

- **Loading Input Image Stack**

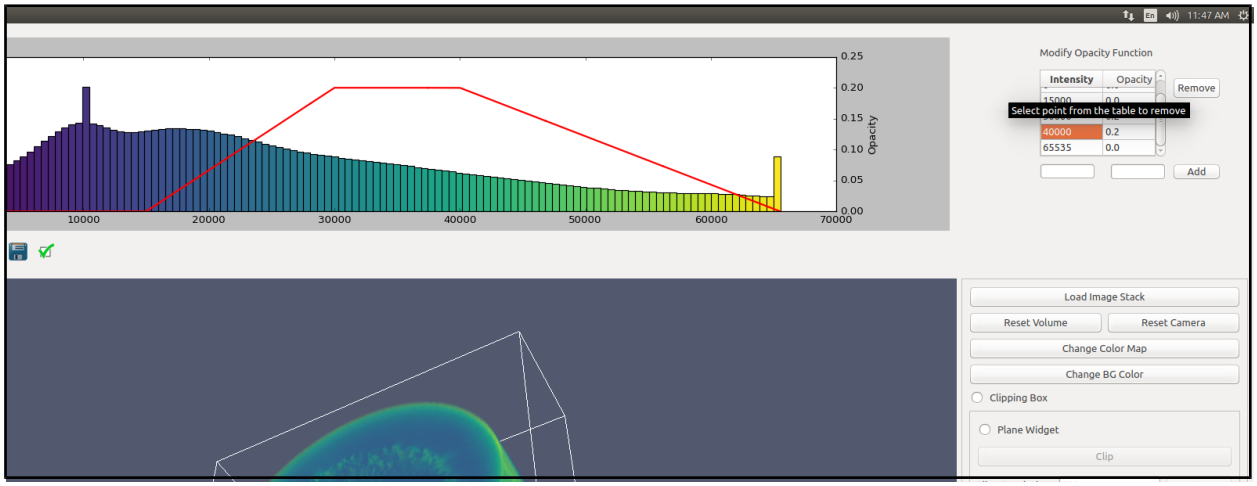


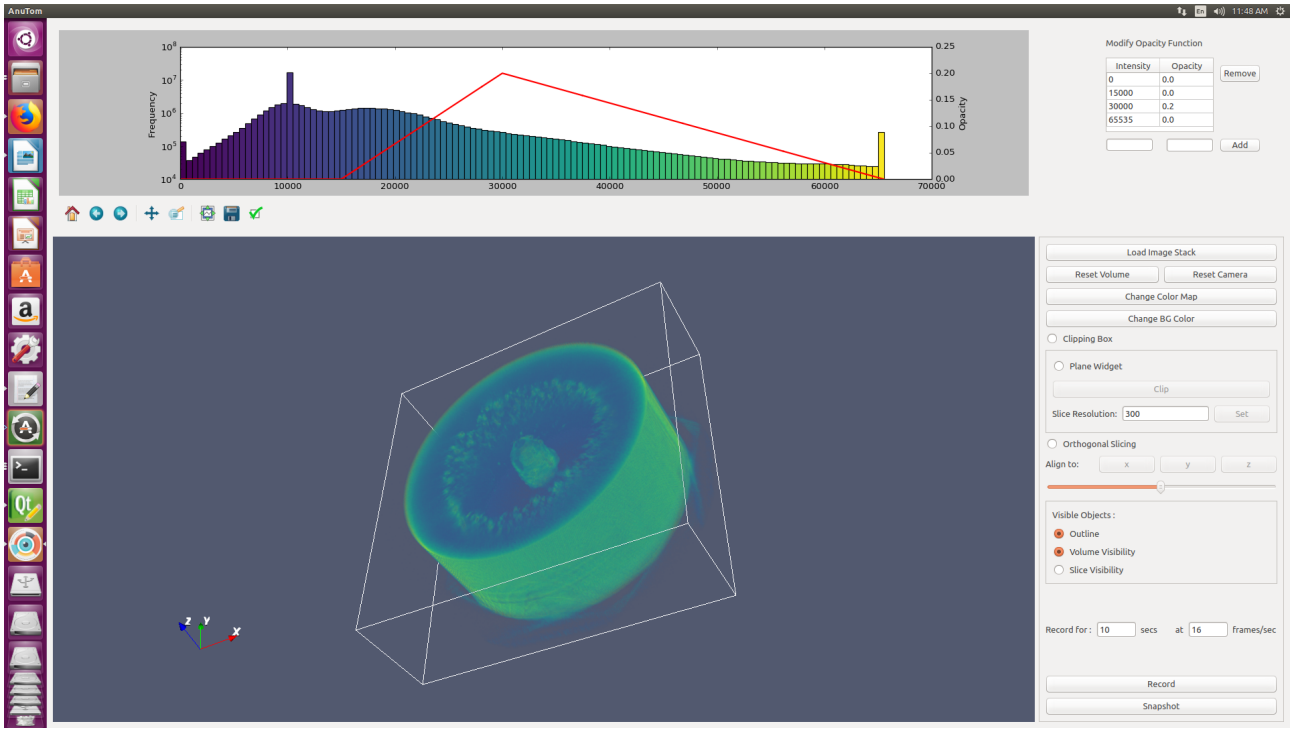
- **Add Alpha function points**



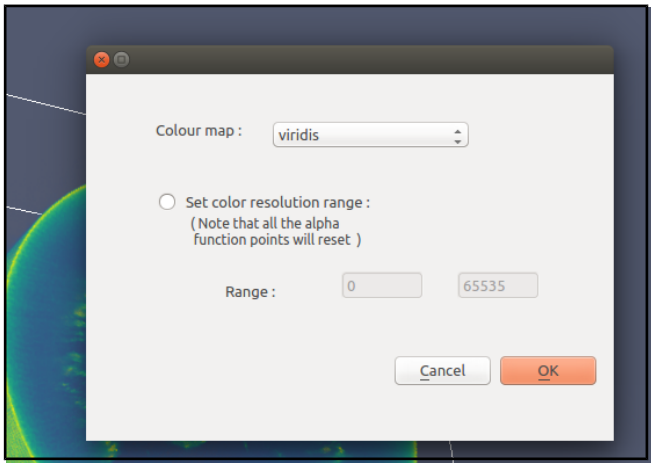


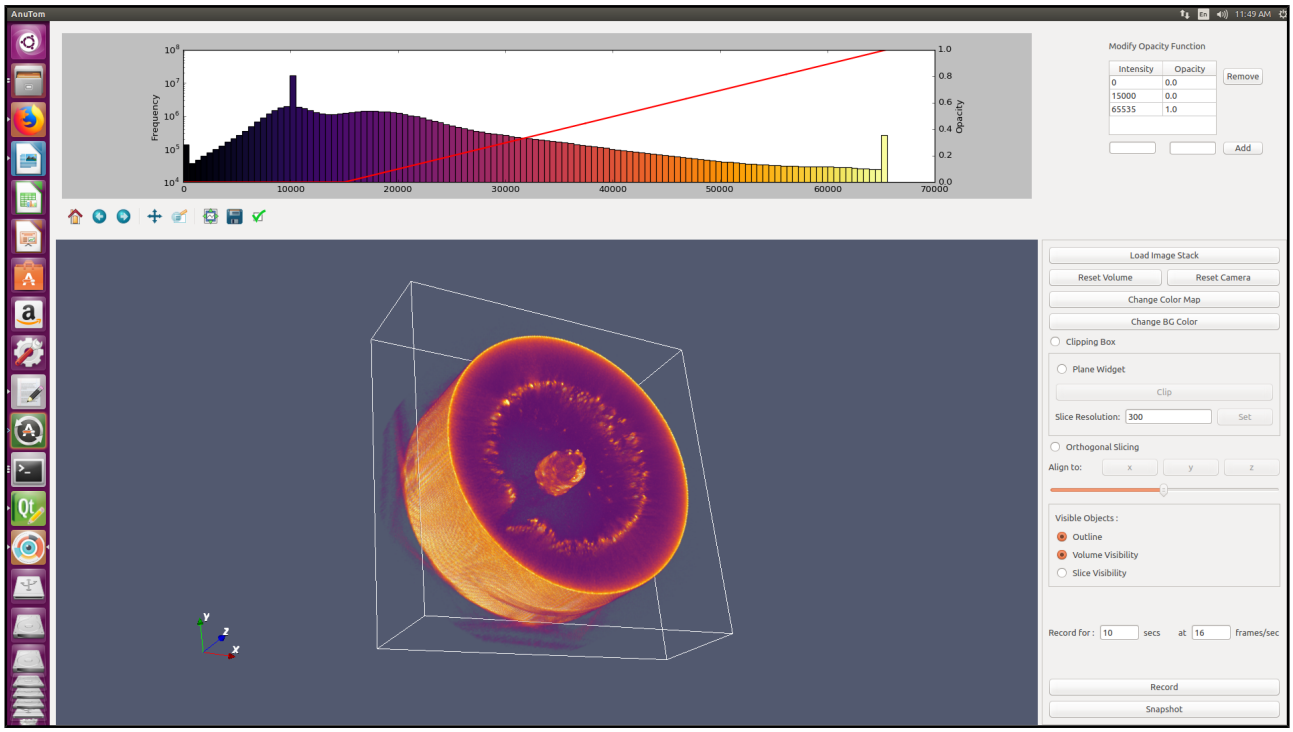
- **Remove Alpha points**





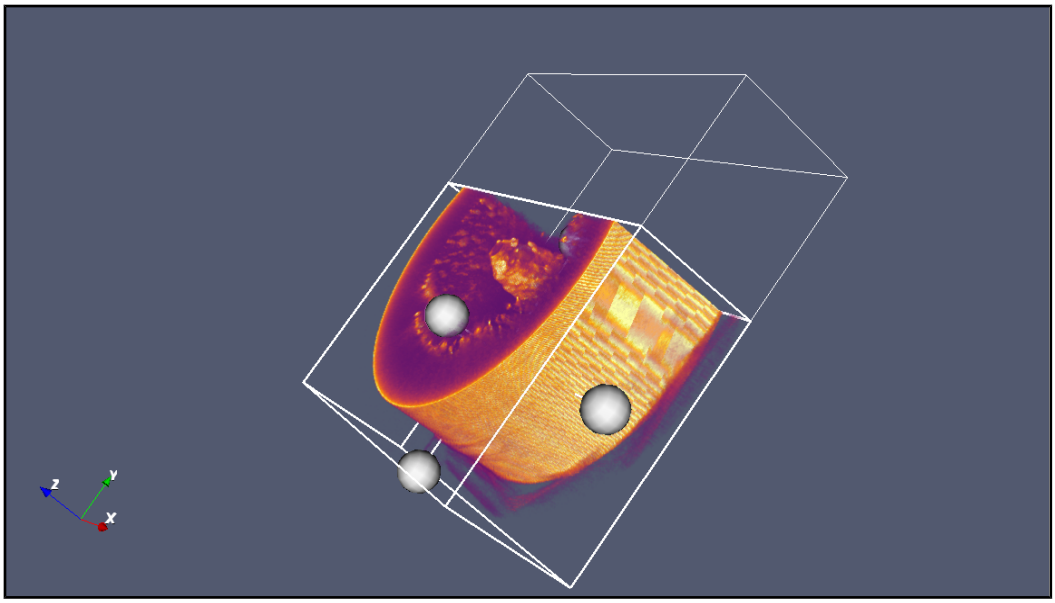
- **Change Color Map**

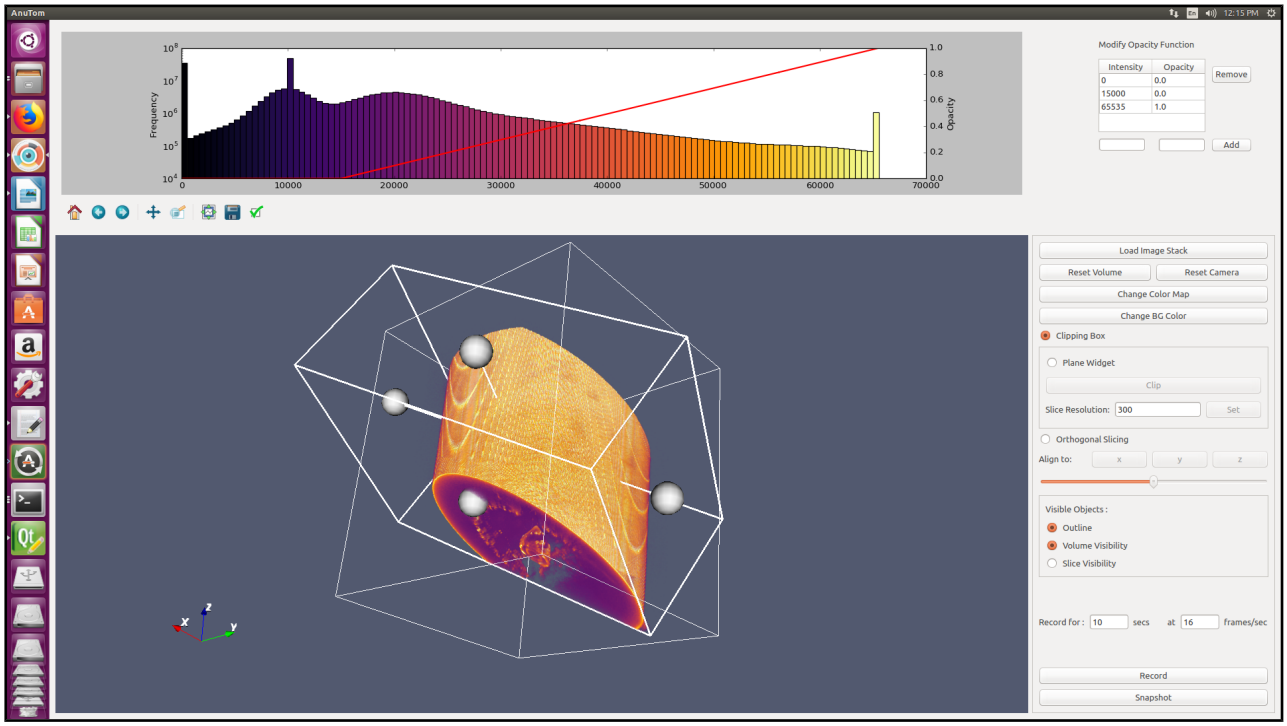




- **Clipping Box**

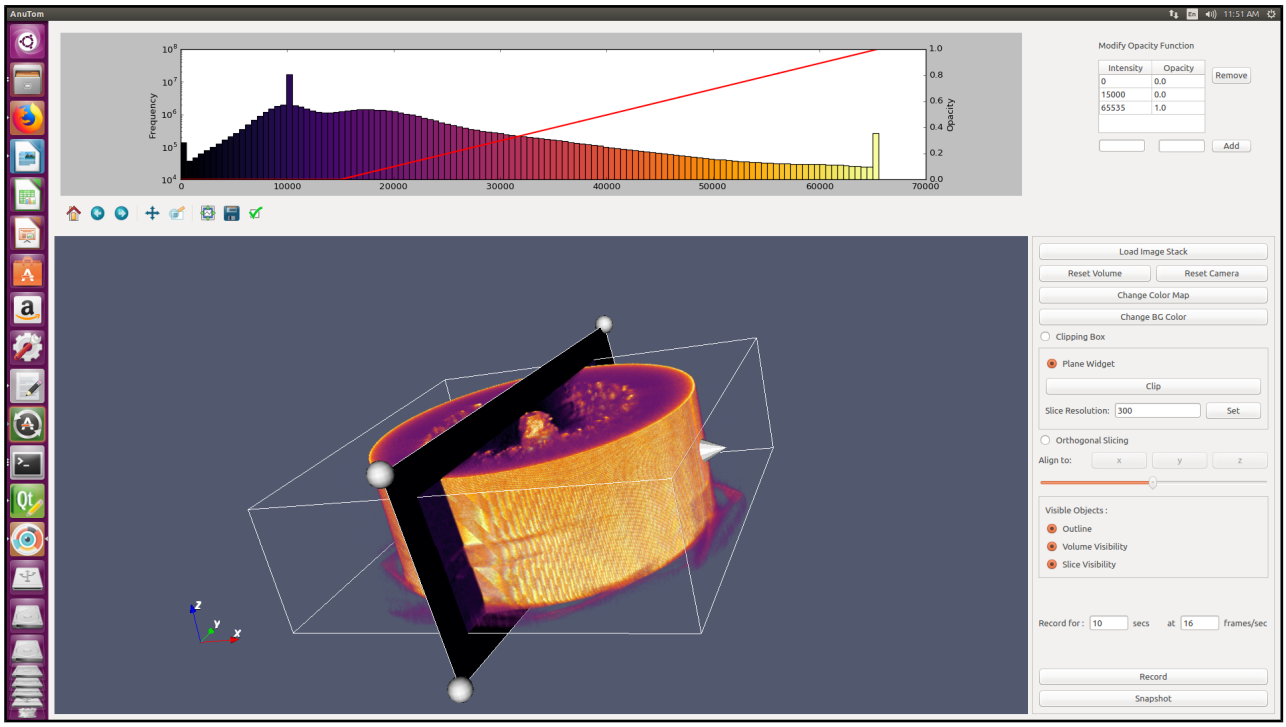
  Multiple plane orthogonal clipping.



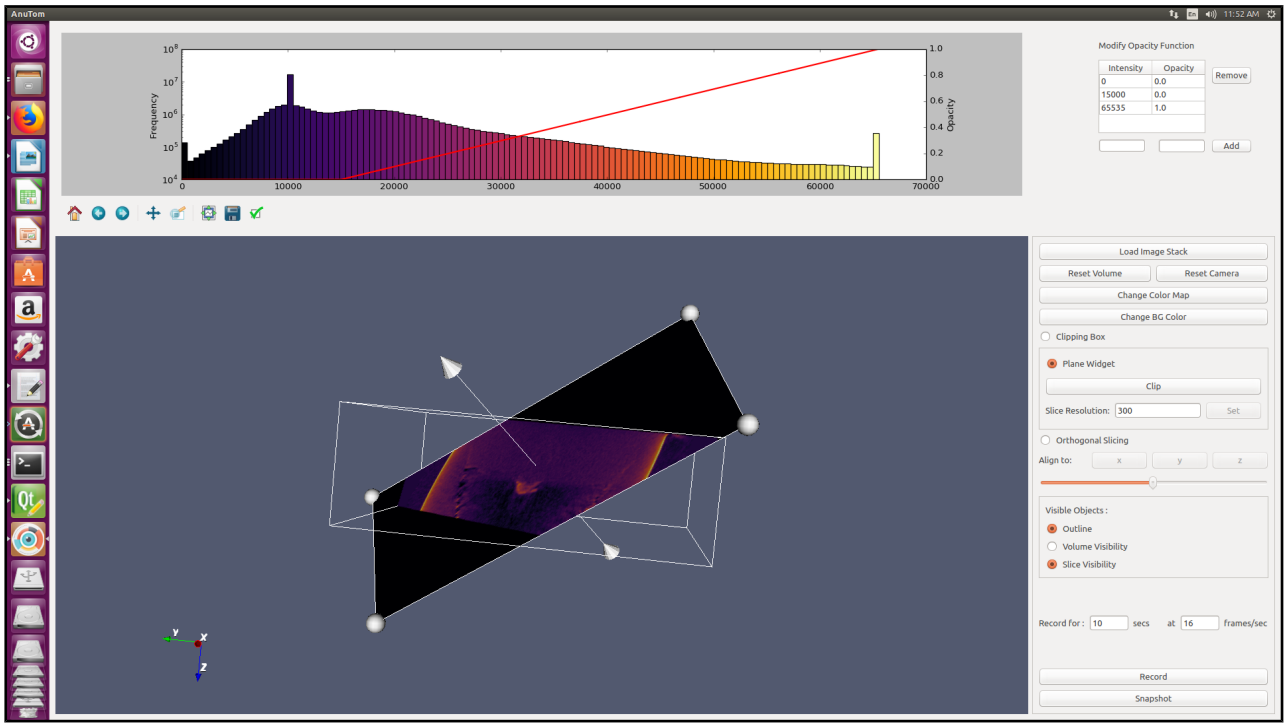


The box can be scaled and rotated in any direction .

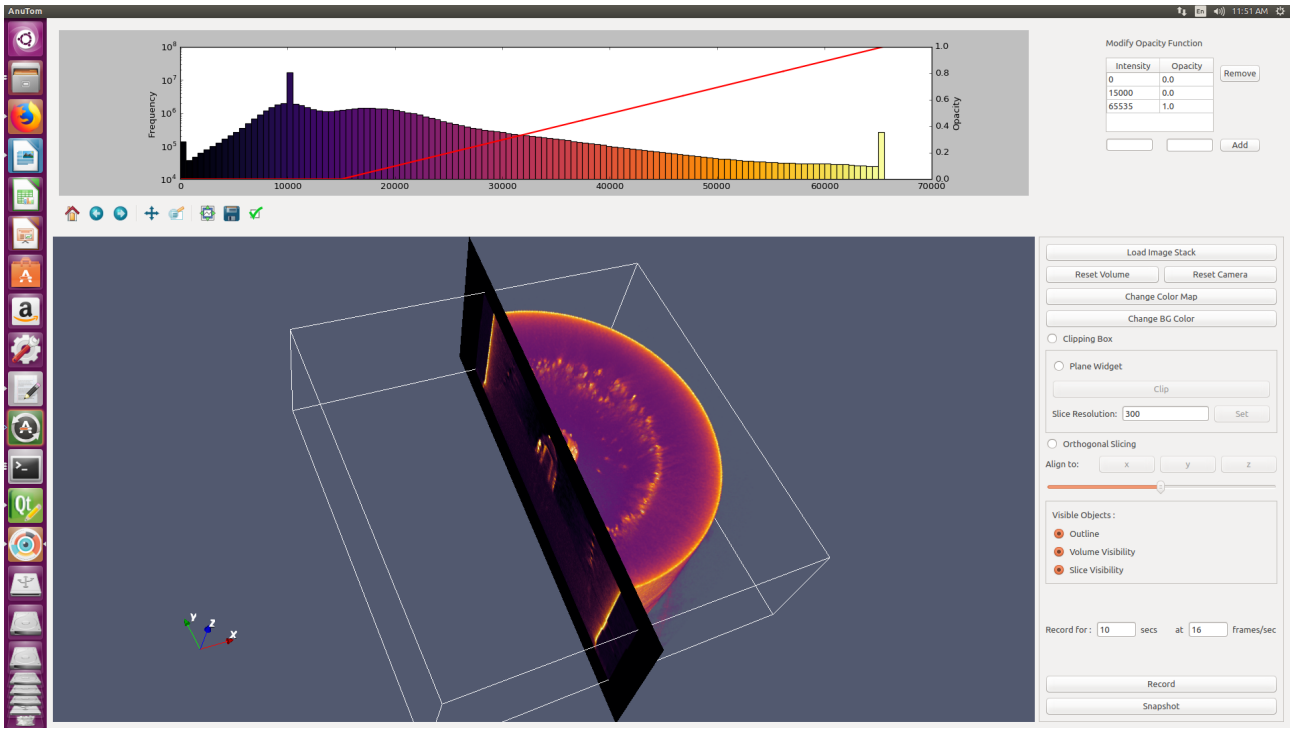
- **Plane Widget**



For Arbitrary Slicing: without any restriction of aligning to any axis

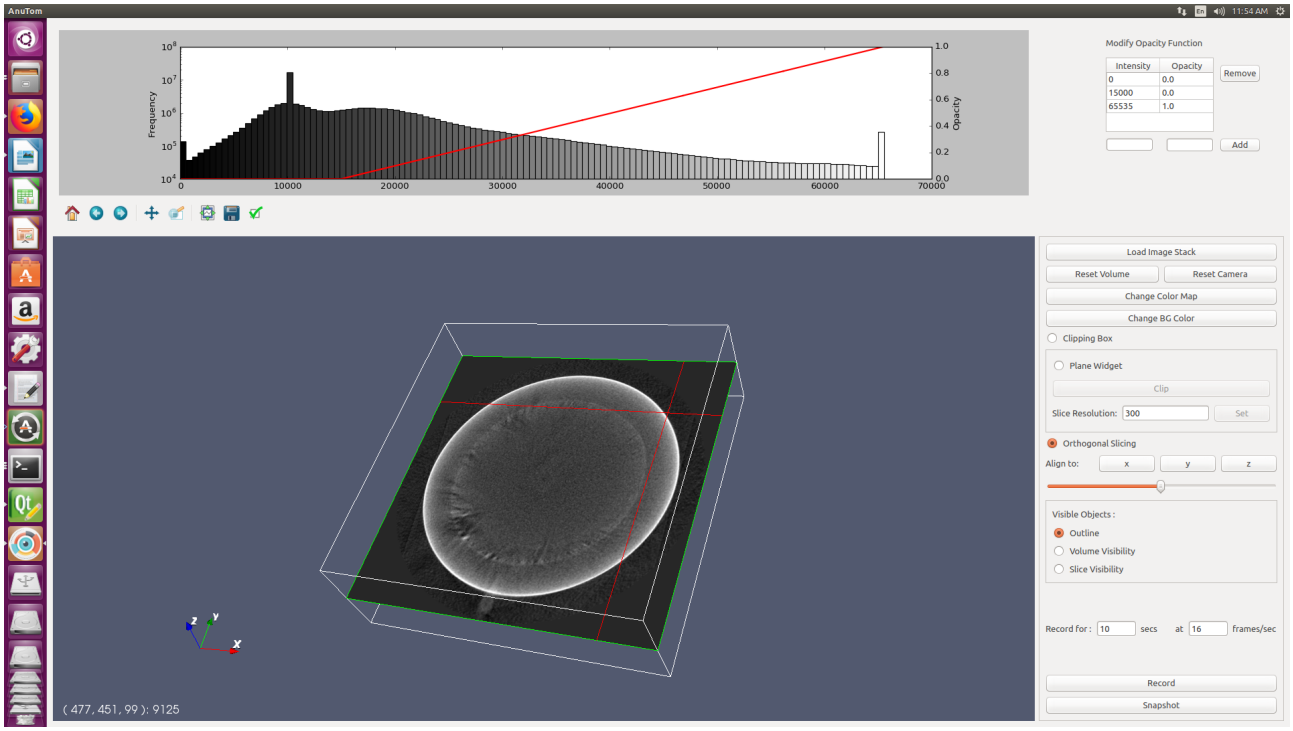


For clipping

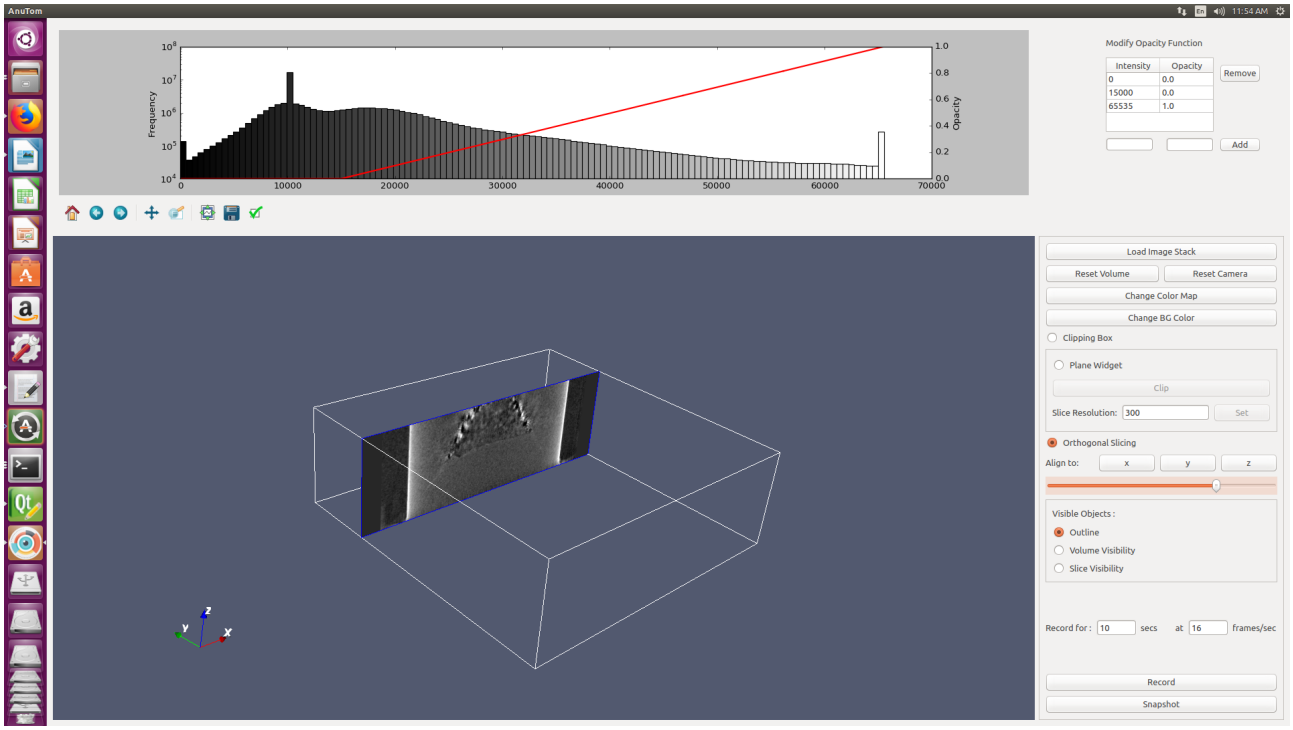


The visibilities of the objects could also be toggled according to user's need

- **Orthogonal Slicing**



The gray value at any point could be seen by clicking on the plane.


AnuTom
Frequency
Opacity
Modify Opacity Function
Intensity
Opacity
Remove
Add
Load Image Stack
Reset Volume
Reset Camera
Change Color Map
Change BG Color
Clipping Box
Plane Widget
Clip
Slice Resolution:
Set
Orthogonal Slicing
Align to:
Visible Objects :
Outline
Volume Visibility
Slice Visibility
Record for :
secs
at
frames/sec
Record
Snapshot

The orientation and position of slice could be changed using buttons and slider

- **Snapshot**

Snapshot of current render window could be taken and saved and also recording the interaction in form of image sequence.