```
#!/usr/bin/env python
# coding: utf-8
```

implementation of volumetric data rendering and 3D visualisation

```
# In[ ]:
#import necessary  libraries
import cv2
import os
import pandas as pd
import numpy as np
import vtk
import matplotlib as mtp
import stl
import webcolors
from ipywidgets import FloatSlider, ColorPicker, VBox, jslink
import ipyvolume as ipv
import numpy as np
import array as arr

# In[ ]:
colors = vtk.vtkNamedColors()
colors.SetColor('SkinColor', [240, 184, 160, 255])
colors.SetColor('BkgColor', [51, 77, 102, 255])

reader=vtk.vtkImageReader()
reader.SetDataScalarType(vtk.VTK_UNSIGNED_SHORT)  # unsigned int16
#reader.SetDataScalarType(vtk.VTK_UNSIGNED_CHAR)  # unsigned int8
```

```
reader.SetFileName(r'C:\Users\hp\Desktop\6th sem\Minor project\Rawdata\3D image Raw data\mecanix\dataset.raw')

reader.SetNumberOfScalarComponents(1)

reader.SetFileDimensionality(3)

reader.SetDataByteOrderToLittleEndian()

reader.SetDataExtent(0,511,0,511,0,742)  #  mage size 512*512*743

reader.SetDataSpacing(1.0, 1.0, 1.0)  # Volume Pixel

reader.Update()

threshold = vtk.vtkImageThreshold()

threshold.SetInputConnection(reader.GetOutputPort())

threshold.ThresholdByLower(20)  #th

threshold.ReplaceInOn()

threshold.SetInValue(1)  # set all values below th to 0

threshold.ReplaceOutOn()

threshold.SetOutValue(2)  # set all values above th to 1

threshold.Update()


volume_mapper = vtk.vtkFixedPointVolumeRayCastMapper()

volume_mapper.SetInputConnection(reader.GetOutputPort())

# The color transfer function maps voxel intensities to colors.

# It is modality-specific, and often anatomy-specific as well.

# The goal is to one color for flesh (between 500 and 1000)

# and another color for bone (1150 and over).


#The 1D transfer function

volume_color = vtk.vtkColorTransferFunction()

volume_color.AddRGBPoint(0, 0.0, 0.0, 0.0)

volume_color.AddRGBPoint(500, 240.0 / 255.0, 184.0 / 255.0, 160.0 / 255.0)

volume_color.AddRGBPoint(1000, 240.0 / 255.0, 184.0 / 255.0, 160.0 / 255.0)

volume_color.AddRGBPoint(1150, 1.0, 1.0, 240.0 / 255.0)  # Ivory
```

```
# The opacity transfer function is used to control the opacity
# of different tissue types.
volume_scalar_opacity = vtk.vtkPiecewiseFunction()
volume_scalar_opacity.AddPoint(0, 0.00)
volume_scalar_opacity.AddPoint(500, 0.15)
volume_scalar_opacity.AddPoint(1000, 0.15)
volume_scalar_opacity.AddPoint(1150, 0.85)

# The gradient opacity function is used to decrease the opacity
# in the 'flat' regions of the volume while maintaining the opacity

volume_gradient_opacity = vtk.vtkPiecewiseFunction()
volume_gradient_opacity.AddPoint(0, 0.0)
volume_gradient_opacity.AddPoint(90, 0.5)
volume_gradient_opacity.AddPoint(100, 1.0)

# The VolumeProperty attaches the color and opacity functions to the
# volume, and sets other volume properties.
volume_property = vtk.vtkVolumeProperty()
volume_property.SetColor(volume_color)
volume_property.SetScalarOpacity(volume_scalar_opacity)
volume_property.SetGradientOpacity(volume_gradient_opacity)
volume_property.SetInterpolationTypeToLinear()
volume_property.ShadeOn()
volume_property.SetAmbient(0.4)
volume_property.SetDiffuse(0.6)
volume_property.SetSpecular(0.2)
# The vtkVolume is a vtkProp3D (like a vtkActor) and controls the position
# and orientation of the volume in world coordinates.
volume = vtk.vtkVolume()
volume.SetMapper(volume_mapper)
```

```
volume.SetProperty(volume_property)

contour=vtk.vtkMarchingCubes()  # vtk.vtkContourFilter()
contour.SetInputConnection(reader.GetOutputPort())
contour.ComputeNormalsOn()
contour.SetValue(0,1)
mapper = vtk.vtkPolyDataMapper()
mapper.SetInputConnection(contour.GetOutputPort())
mapper.ScalarVisibilityOff()
actor = vtk.vtkActor()
actor.SetMapper(mapper)
actor.GetProperty().SetColor(1.0,0.0,0.0)
actor.GetProperty().SetOpacity( 0.0 )

contourBoneHead = vtk.vtkMarchingCubes()
contourBoneHead.SetInputConnection( reader.GetOutputPort() )

contourBoneHead.ComputeNormalsOn()

contourBoneHead.SetValue( 0, 1250 )  # Bone isovalue


# Take the isosurface data and create geometry

geoBoneMapper = vtk.vtkPolyDataMapper()

geoBoneMapper.SetInputConnection( contourBoneHead.GetOutputPort() )

geoBoneMapper.ScalarVisibilityOff()
```

```
# Take the isosurface data and create geometry

actorBone = vtk.vtkLODActor()

actorBone.SetNumberOfCloudPoints( 1000000 )

actorBone.SetMapper( geoBoneMapper )

actorBone.GetProperty().SetColor( 1, 1, 1 )
actorBone.GetProperty().SetOpacity(0.8)
skin_extractor = vtk.vtkMarchingCubes()
skin_extractor.SetInputConnection(reader.GetOutputPort())
skin_extractor.SetValue(0, 500)
skin_extractor.Update()

skin_stripper = vtk.vtkStripper()
skin_stripper.SetInputConnection(skin_extractor.GetOutputPort())
skin_stripper.Update()

skin_mapper = vtk.vtkPolyDataMapper()
skin_mapper.SetInputConnection(skin_stripper.GetOutputPort())
skin_mapper.ScalarVisibilityOff()

skin = vtk.vtkActor()
skin.SetMapper(skin_mapper)
skin.GetProperty().SetDiffuseColor(colors.GetColor3d('SkinColor'))
skin.GetProperty().SetSpecular(0.3)
skin.GetProperty().SetSpecularPower(20)
skin.GetProperty().SetOpacity(0.5)
bone_extractor = vtk.vtkMarchingCubes()
bone_extractor.SetInputConnection(reader.GetOutputPort())
```

```
bone_extractor.SetValue(0,1150)

bone_stripper = vtk.vtkStripper()
bone_stripper.SetInputConnection(bone_extractor.GetOutputPort())

bone_mapper = vtk.vtkPolyDataMapper()
bone_mapper.SetInputConnection(bone_stripper.GetOutputPort())
bone_mapper.ScalarVisibilityOff()

bone = vtk.vtkActor()
bone.SetMapper(bone_mapper)
bone.GetProperty().SetDiffuseColor(colors.GetColor3d('Ivory'))

# An outline provides context around the data.
#
outline_data = vtk.vtkOutlineFilter()
outline_data.SetInputConnection(reader.GetOutputPort())
outline_data.Update()

map_outline = vtk.vtkPolyDataMapper()
map_outline.SetInputConnection(outline_data.GetOutputPort())

outline = vtk.vtkActor()
outline.SetMapper(map_outline)
outline.GetProperty().SetColor(colors.GetColor3d('Black'))
outline_data = vtk.vtkOutlineFilter()

# Now we are creating three orthogonal planes passing through the
# volume. Each plane uses a different texture map and therefore has
# different coloration.
```

```
# Start by creating a black/white lookup table.
bw_lut = vtk.vtkLookupTable()
bw_lut.SetTableRange(0, 2000)
bw_lut.SetSaturationRange(0, 0)
bw_lut.SetHueRange(0, 0)
bw_lut.SetValueRange(0, 1)
bw_lut.Build()  # effective built

# Now create a lookup table that consists of the full hue circle
# (from HSV).
hue_lut = vtk.vtkLookupTable()
hue_lut.SetTableRange(0, 2000)
hue_lut.SetHueRange(0, 1)
hue_lut.SetSaturationRange(1, 1)
hue_lut.SetValueRange(1, 1)
hue_lut.Build()  # effective built

# Finally, create a lookup table with a single hue but having a range
# in the saturation of the hue.
sat_lut = vtk.vtkLookupTable()
sat_lut.SetTableRange(0, 2000)
sat_lut.SetHueRange(0.6, 0.6)
sat_lut.SetSaturationRange(0, 1)
sat_lut.SetValueRange(1, 1)
sat_lut.Build()  # effective built

# Create the first of the three planes. The filter vtkImageMapToColors
# maps the data through the corresponding lookup table created above.  The

sagittal_colors = vtk.vtkImageMapToColors()
sagittal_colors.SetInputConnection(reader.GetOutputPort())
```

```
sagittal_colors.SetLookupTable(bw_lut)
sagittal_colors.Update()

sagittal = vtk.vtkImageActor()
sagittal.GetMapper().SetInputConnection(sagittal_colors.GetOutputPort())
sagittal.SetDisplayExtent(270, 270, 0, 530, 0, 760)
sagittal.ForceOpaqueOn()

# Create the second (axial) plane of the three planes. We use the
# same approach as before except that the extent differs.
axial_colors = vtk.vtkImageMapToColors()
axial_colors.SetInputConnection(reader.GetOutputPort())
axial_colors.SetLookupTable(hue_lut)
axial_colors.Update()

axial = vtk.vtkImageActor()
axial.GetMapper().SetInputConnection(axial_colors.GetOutputPort())
axial.SetDisplayExtent(0, 530, 0, 530, 390, 390)
axial.ForceOpaqueOn()

# Create the third (coronal) plane of the three planes. We use
# the same approach as before except that the extent differs.
coronal_colors = vtk.vtkImageMapToColors()
coronal_colors.SetInputConnection(reader.GetOutputPort())
coronal_colors.SetLookupTable(sat_lut)
coronal_colors.Update()

coronal = vtk.vtkImageActor()
coronal.GetMapper().SetInputConnection(coronal_colors.GetOutputPort())
coronal.SetDisplayExtent(0, 530, 270, 270, 0, 760)
coronal.ForceOpaqueOn()
```

```
a_camera = vtk.vtkCamera()
a_camera.SetViewUp(0, 0, 1)
a_camera.SetPosition(0, 1, 0)
a_camera.SetFocalPoint(0, 0, 0)
a_camera.ComputeViewPlaneNormal()
a_camera.Azimuth(30.0)
a_camera.Elevation(30.0)



# Actors are added to the renderer.

renderer=vtk.vtkRenderer()
renderer.SetBackground([0.329412, 0.34902, 0.427451])
renderer.AddActor(actor)
renderer.AddActor(actorBone)
renderer.AddActor(skin)
renderer.AddActor(bone)
renderer.AddActor(outline)
renderer.AddActor(sagittal)
renderer.AddActor(axial)
renderer.AddActor(coronal)
renderer.AddViewProp(volume)

# Turn off bone for this example.
bone.VisibilityOff()

   # Set skin to semi-transparent.
renderer.SetActiveCamera(a_camera)
```

create renderer to render the window

In[ ]:

```
    renderer.ResetCamera()
    a_camera.Dolly(1.5)
    window = vtk.vtkRenderWindow()
    window.SetSize(640, 640)
    window.AddRenderer(renderer)
    window.SetWindowName('Medical Image Visualisation')
```

calling the transfer function

In[ ]:

```
from ipyTransferFunction import TransferFunctionEditor

def display_palette_info(reader):
    print(reader.data_range)

tf = TransferFunctionEditor(
    name='rainbow', size=32, alpha=0.5,
    continuous_update=False, on_change=renderer)
```

In[ ]:

```
tf.set_palette('seismic')
tf.set_range((0,255))
```

create window interactor to interact

In[ ]:

```
interactor = vtk.vtkRenderWindowInteractor()
interactor.SetRenderWindow(window)

renderer.ResetCameraClippingRange()

window.Render()
interactor.Initialize()
interactor.Start()
```

#transfer function

```
import seaborn as sns
from ipywidgets import widgets, Layout, Box, VBox, ColorPicker
from IPython.display import display
```

```
class TransferFunctionEditor(object):

    def __init__(self, filename=None, name='rainbow',
                 size=32, alpha=0.0, data_range=(0, 255),
                 continuous_update=False, on_change=None):
        self.palette = list()
        self.alpha_sliders = list()
        self.color_pickers = list()
        self.continuous_update = continuous_update
        self.data_range = data_range
        self.send_updates_to_renderer = True
        self._on_change = on_change

        if filename is None:
            # Initialize palette from seaborn
            self.palette.clear()
            for color in sns.color_palette(name, size):
                self.palette.append([color[0], color[1], color[2], alpha])
        else:
            # Load palette from file
            self._load(filename)

        # Create control and assign palette
        self._create_controls()
        self._update_controls()
        self._callback()

    def _load(self, filename):
        # Clear controls
```

```
    self.alpha_sliders.clear()
    self.color_pickers.clear()

    # Read colormap file
    lines = tuple(open(filename, 'r'))
    self.palette.clear()
    for line in lines:
        words = line.split()
        if len(words) == 4:
            r = float(words[0])
            g = float(words[1])
            b = float(words[2])
            a = float(words[3])
            color = [r, g, b, a]
            self.palette.append(color)

def save(self, filename):
    with open(filename, 'w') as f:
        f.write(str(len(self.palette)) + '\n')
        for color in self.palette:
            f.write(str(color[0]) + ' ' + str(color[1]) + ' ' + str(color[2]) + ' ' + str(color[3]) + '\n')
        f.close()

def set_palette(self, name):
    size = len(self.palette)
    newPalette = sns.color_palette(name, size)
    for i in range(size):
        color = newPalette[i]
        self.palette[i] = [color[0], color[1], color[2], self.palette[i][3]]
    self._update_controls()
    self._callback()
```

```
def set_range(self, range):
    self.data_range = range
    self._callback()

def _html_color(self, index):
    color = self.palette[index]
    color_as_string = '#' \
                '%02x' % (int)(color[0] * 255) + \
                '%02x' % (int)(color[1] * 255) + \
                '%02x' % (int)(color[2] * 255)
    return color_as_string

def _update_colormap(self, change):
    self._callback()

def _update_colorpicker(self, change):
    for i in range(len(self.palette)):
        self.alpha_sliders[i].style.handle_color = self.color_pickers[i].value
    self._callback()

def _create_controls(self):
    self.send_updates_to_renderer = False
    # Layout
    alpha_slider_item_layout = Layout(
        overflow_x='hidden', height='180px', max_width='20px')
    color_picker_item_layout = Layout(
        overflow_x='hidden', height='20px', max_width='20px')
    box_layout = Layout(display='inline-flex')

    # Sliders
```

```
    self.alpha_sliders = [widgets.IntSlider(
        continuous_update=self.continuous_update,
        layout=alpha_slider_item_layout,
        description=str(i),
        orientation='vertical',
        readout=True,
        value=self.palette[i][3] * 256, min=0, max=255, step=1
    ) for i in range(len(self.palette))]

    # Color pickers
    self.color_pickers = [
        ColorPicker(
            layout=color_picker_item_layout,
            concise=True,
            disabled=False) for i in range(len(self.palette))
    ]
    # Display controls
    color_box = Box(children=self.color_pickers)
    alpha_box = Box(children=self.alpha_sliders)
    box = VBox([color_box, alpha_box], layout=box_layout)

    # Attach observers
    for i in range(len(self.palette)):
        self.alpha_sliders[i].observe(self._update_colormap, names='value')
        self.color_pickers[i].observe(self._update_colorpicker, names='value')
    display(box)
    self.send_updates_to_renderer = True

def _update_controls(self):
    self.send_updates_to_renderer = False
    for i in range(len(self.palette)):
```

```
            color = self._html_color(i)
            self.alpha_sliders[i].style.handle_color = color
            self.color_pickers[i].value = color
        self.send_updates_to_renderer = True

    def _callback(self):
        from webcolors import name_to_rgb, hex_to_rgb

        if not self.send_updates_to_renderer:
            return

        for i in range(len(self.palette)):
            try:
                color = name_to_rgb(self.color_pickers[i].value)
            except ValueError:
                color = hex_to_rgb(self.color_pickers[i].value)

            c = [
                float(color.red) / 255.0,
                float(color.green) / 255.0,
                float(color.blue) / 255.0,
                float(self.alpha_sliders[i].value) / 255.0
            ]
            self.palette[i] = c

        if self._on_change:
            self._on_change(self)
```