Image Processing Homework 8

吴嘉骜 21307130203

2023 年 12 月 24 日

1 Problem 1

Invoke the Visualization Toolkit (VTK) library to implement a complete rendering process for 3D volumetric data, including aspects such as lighting models and color settings.
Note: Choose one of the two rendering processes to implement: (1) Isosurface rendering, or (2) Volume rendering.
Solution:
We choose to implement volume rendering. See the full codes in `volrender.py`. The volume data for rendering is `image_lr.nii.gz`, as provided. To load the data, we use the `nibabel` library, which is a Python interface to the NIfTI file format. Here we only show the main steps of the program.

1.1 VTK Image Data Setup

```
vtk_image = vtk.vtkImageData()
vtk_image.SetDimensions(dims[0], dims[1], dims[2])
vtk_image.SetSpacing(spacing[0], spacing[1], spacing[2])
vtk_image.SetOrigin(0, 0, 0)
```

Here, a `vtkImageData` object is created and configured with the dimensions and spacing obtained from the medical image. This object serves as the container for the image data in the VTK pipeline.

1.2 Filling the VTK Image with Scalar Data

```
int_range = (20, 500) # Set the intensity range
max_u_short = 128 # Set the maximum value for unsigned short
vtk_image.AllocateScalars(vtk.VTK_DOUBLE, 1)
for z in range(dims[2]):
    for y in range(dims[1]):
        for x in range(dims[0]):
            scalar_data = img_data[x, y, z]
            if scalar_data < int_range[0]:
                scalar_data = int_range[0]
            if scalar_data > int_range[1]:
                scalar_data = int_range[1]
```

```
            scalar_data = max_u_short*float(scalar_data-int_range[0])/float(int_range[1]-int_range[0])
            vtk_image.SetScalarComponentFromDouble(x, y, z, 0, scalar_data)
```

This section scales and sets the scalar data within the specified intensity range for the VTK image. The data is normalized to fit within the range of an unsigned short type.

1.3 Defining Transfer Function

```
opacity_function = vtk.vtkPiecewiseFunction()
opacity_function.AddSegment(0, 0, 10, 0)
opacity_function.AddSegment(10, 0.2, 120, 0.2)
volume_property.SetScalarOpacity(opacity_function)

color_function = vtk.vtkColorTransferFunction()
color_function.AddRGBSegment(0, 0, 0, 0, 20, 0.2, 0.2, 0.2)
color_function.AddRGBSegment(20, 0.1, 0.2, 0, 128, 1, 1, 0)
volume_property.SetColor(color_function)

gradient_opacity = vtk.vtkPiecewiseFunction()
gradient_opacity.AddPoint(0, 0.0)
gradient_opacity.AddSegment(27, 0.3, 128, 0.4)
volume_property.SetGradientOpacity(gradient_opacity)
```

This first sets up piecewise functions for opacity and color respectively, defining how transparent the volume will appear at different intensity values and how different intensity values within the volume will be mapped to specific colors. The gradient opacity function is then used to adjust opacity based on the rate of change in intensity, enhancing edges and fine details.

1.4 Optical Properties Configuration

```
volume_property.SetInterpolationTypeToLinear()
volume_property.SetAmbient(1)
volume_property.SetDiffuse(0.9)
volume_property.SetSpecular(0.6)
volume_property.SetSpecularPower(10)
```

Here, various optical properties like ambient light, diffusion, and specular highlights are configured for the volume.

1.5 Rendering

```
renderer = vtk.vtkRenderer()
renderer.SetBackground(1, 1, 1)
renderer.AddVolume(volume)
light = vtk.vtkLight()
```

```
light.SetColor(0, 1, 1)
renderer.AddLight(light)
render_window = vtk.vtkRenderWindow()
render_window.AddRenderer(renderer)
render_window.SetSize(500, 500)
interactor = vtk.vtkRenderWindowInteractor()
interactor.SetRenderWindow(render_window)
interactor.Initialize()
render_window.Render()
interactor.Start()
```

This section initializes the renderer, render window and interactor, and then starts the rendering process. Note that a light source is added to the renderer for enhanced visualization.

The result is shown in Figure 1. It is a volume rendering of the heart CT image, and the heart is clearly visible despite some fragments around it. It is recommended to run the program and use the mouse to rotate the image for better observation.

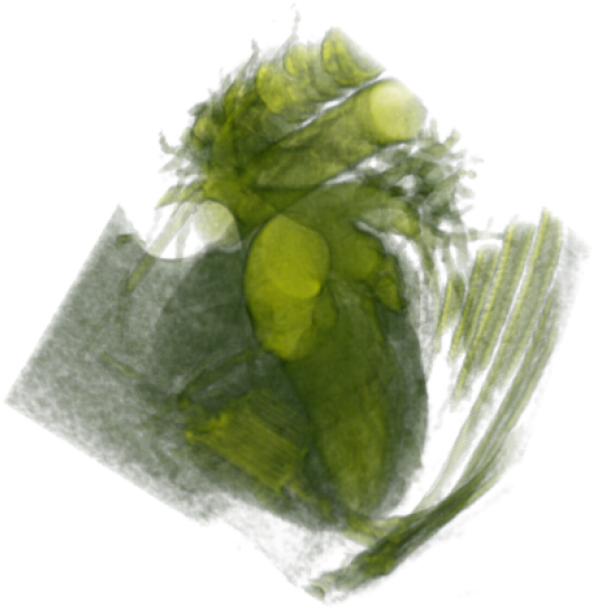


Figure 1: Volume rendering result

2 Problem 2

Please design a method to eliminate the fragmented surface elements in the isosurface rendering result of a heart CT image (image_lr.nii.gz).

Note: Choose one of the two ways: (1) Describe the solution in words; (2) Or implement and demonstrate the result using code.

Solution:

We utilize the 3D sliding average method to smooth the image, and then apply the marching cubes algorithm to generate the isosurface. See the original isosurface rendering result by running `surrender.py`,

and the denoised figure by running `denoise.py`.

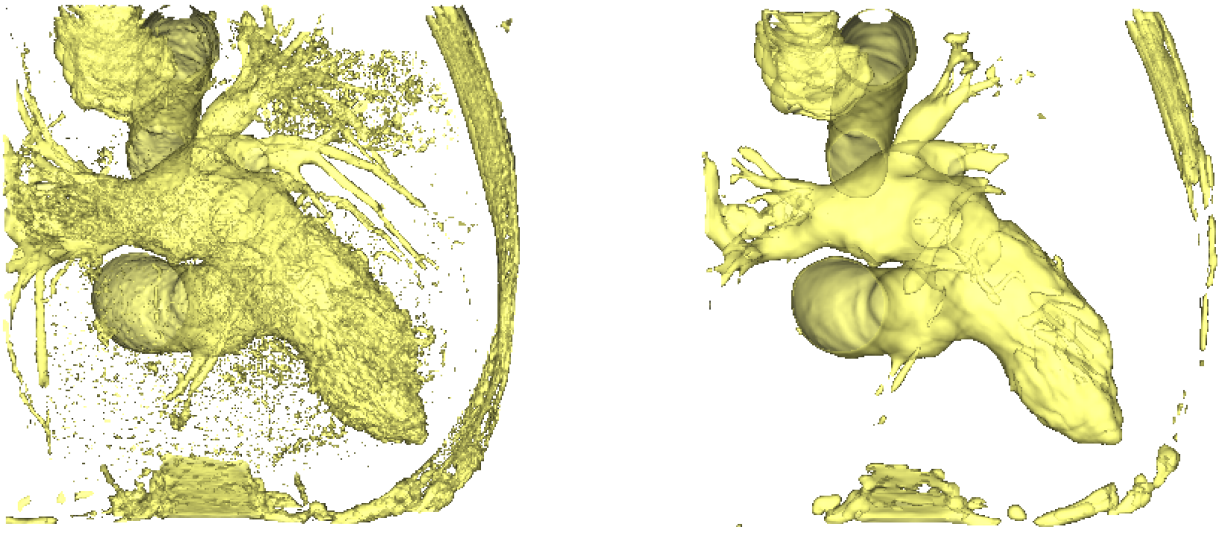
Below is the main algorithm for denoising.

```
# 3D Zero-padding
nwin = 3
halfwin = int(nwin / 2)
padded_img = np.pad(img_data, halfwin, mode='constant')
print("Zero-padding complete.")

# Apply a 3D sliding average for smoothing
kernel = np.ones((nwin, nwin, nwin)) / (nwin**3)
smoothed_img = np.zeros_like(img_data)

for i in range(dims[0]):
    for j in range(dims[1]):
        for k in range(dims[2]):
            smoothed_img[i, j, k] = np.sum(
                padded_img[i:i+nwin, j:j+nwin, k:k+nwin] * kernel)
print("3D smoothing processing complete.")
```

We define a $3 \times 3 \times 3$ kernel and apply it to the image using a sliding window. The kernel is a matrix of ones divided by the number of elements in the matrix, which is equivalent to averaging the values in the window. Before applying the kernel, we pad the image with zeros to avoid the sliding window going out of bounds. The result is shown in Figure 2.



(a) Original isosurface rendering result

(b) Denoised isosurface rendering result

Figure 2: Isosurface rendering denoise

We can see that the fragmented surface elements are eliminated, and the heart is more clearly visible. However, some details at the end of the heart are lost in the process, indicating that a more sophisticated denoising method may be needed.