Topic 8: Volume Visualization with Raycasting and Transfer Function

NAME: XIYUAN WANG STUDENT NO. 2018533177 EMAIL:WANGXY7@SHANGHAITECH.EDU.CN
NAME: CHUYI ZHAO STUDENT NO. 2018533130 EMAIL: ZHAOCHY1@SHANGHAITECH.EDU.CN
NAME: TAOTAO ZHOU STUDENT NO. 2019533105 EMAIL: ZHAOCHY1@SHANGHAITECH.EDU.CN

1 INTRODUCTION

Scalar field visualization has always received a lot of attention in visualization domain, especially the 3D scalar field visualization, including scientific computing data visualization and engineering computing data visualization. It also includes visualizations of various measurement datasets like CT and MRI data.

Volume rendering is a very important method in visualization algorithms because it can not only display the surface information of a volume data, the internal information can also be displayed clearly as well. Volume rendering transfers a large amount of abstract data into a visual image to enable people to get insight into the hiding information. The visibility and color of the voxel in the volume dataset drawn in the image depends on the opacity value assigned by the transfer function.

2 IMPLEMENTATION DETAILS

The technique of volume ray casting can be derived directly from the rendering equation. It provides results of very high quality rendering. Volume ray casting is classified as an image-based volume rendering technique, as the computation emanates from the output image and not the input volume data. There are four basic steps in this algorithm. Fig.1 shows the general pipeline below.

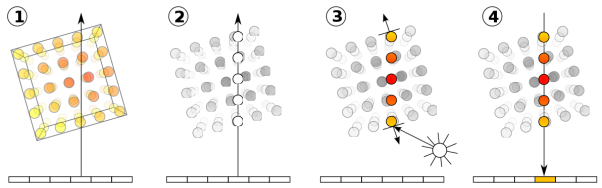

Fig. 1. four basic steps of volume ray casting

2.1 Ray Casting

For a 3D texture, a 3D model is required to contain the texture so that we can sample the data correctly. For each pixel of the final image, a ray of sight is cast through the volume, whose direction is computed by calculating the position of the view point and the pixels of the surface of image sequence.

2.2 Sampling

To sample the 3D volume texture, we employ an equal-interval sampling technique to get the properties of each pixel we sampled. However, the distance between two points does not have to be equal at any time. Typically, we use a 0.001 step size. When the volume data is sparser than our step size, we interpolate the intensity values between the two sample points along the same ray It's important to determine when the ray passes through the volume to stop sampling. For OpenGL, we need to first get the color value of both front depth and back depth, representing the distance nearest and farthest from the view point. Then it's easy to get the distance of ray casting by subtracting front values with back values.

2.3 Shading and compositing

Rendering a transparent object A is actually blending its color with the color of another object B behind it. This method is called alpha blending: $c_0 = a_s + (1 - a_s)c_d$, where a_s is the transparency of A while c_s represents its actual color. c_d is the color of the object B and c_0 is the color we finally get by looking at B through the transparent A. If there are many transparent objects, sorting them is required unless their transparency are all the same. Intuitively, there are two ways to sample and accumulate the color: either from front to back or from back to front. Here we use the first one. For color, the equation is $C_i = (1 - A_{i-1})C_i + C'_{i-1}$, for transparency, is $A_i = (1 - A_{i-1})A_i + A_{i-1}$.Also, we can apply some lighting models like Phong Lighting Model to make the object more realistic.

2.4 Transfer Function

Transfer function is useful to help us visualize data, analysis data and processing data. Normally, a transfer function can map the information of the data to color or opacity values. With the help of rendering functions, we can finally get the image of the data in another color space. There are a bunch of transfer function types. The most commonly used one is which based on scalar values. In this project, the volume data is in .raw format, which is commonly used to store CT scans. Some geometric attributes like normal and curvatures cannot be applied. As a result, we use the intensity values of each voxel as the input of our transfer function.

2.4.1 Design. To better explore the properties of data, we first do a statistical analysis of the given volume data. We use 256 bins, whose corresponding values ranges from 0 to 255. We compute the histogram graph of the raw data and draw it in a window with gray color. Referring to the histogram graph, we can better exploit Our transfer function takes directly the intensity scalar value as input. We plot the intensity histogram and use four channels to adjust the transfer function: red, green, blue and opacity channels, respectively deciding the color and transparency of voxels. Each channel has three parameters(two parameters for opacity channel) to be changed.

2.4.2 UI Interface. We use *glui* to construct an interface menu and *OpenCV2* to plot the intensity histogram and visualize the

(a) (b)

Fig. 2. UI interface

transfer function. Users can adjust the channel state(enabled or not), mean and width through spinning or scrolling. In detail, the intensity scalars are in range of [0, 255]. Therefore, the voxels can be divided into different partitions according to their scalar value. We use a mask with manually set mean and width to choose different partition ans set their color and opacity. The results are rendered and displayed in real-time, showing in the following result part.

3 RESULTS

We implement a visualization interface for Transfer Function designing. Some results are shown in the final pages.
Our application is able to rotate the view of the camera and observe the 3D rendered volume data interactively. There are 3 display windows in pur application. The first one is for displaying the 3D rendered results of our transfer functions. The second one is made up of four slider bars, for RGBA four channels respectively. Also, we have buttons to adjust parameters of the transfer function. By displaying the real-time transfer function, users are able to find the best rendering result that meets their needs. Below are our volume rendering results tested on 5 raw format data. We can vividly observe that, by appying transfer function, we are able to partially distangle the visualization of different parts of an object. The inner bones or outer layer of one object could be colored by different colors easily.

4 CONCLUSION

By applying the ray-casting algorithm, we successfully achieve the objective to render volume data and design transfer functions to visualize them in an elegant way. Volume rendering gives us an intuitive way to look around the inner part or outer part of an object that is not transparent. It perfectly shows the charm of data visualization. An image is more than a thousand words.

5 DISCUSSION

Although it's feasible to add Phong Lighting Model to make the object we are rendering more realistic, we failed to compute the normals of the surface due to lack of geometric information of the raw data. Maybe we can use the intensity values to compute the gradients of every 2D textures and save them previously so as to apply the lighting models.

6 REFERENCE

Due to lack of knowledge for *openGL* usage(which is not covered in this course), we implement our project based on a framework from *github*, which implements process of reading in a raw file and represent it.

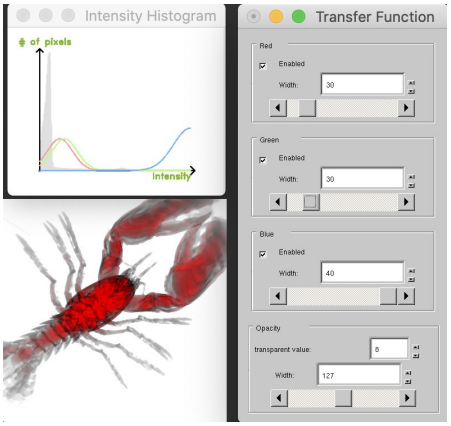

Fig. 4. **By adjusting color and opacity, the texture and detail of lobster are vivid.**

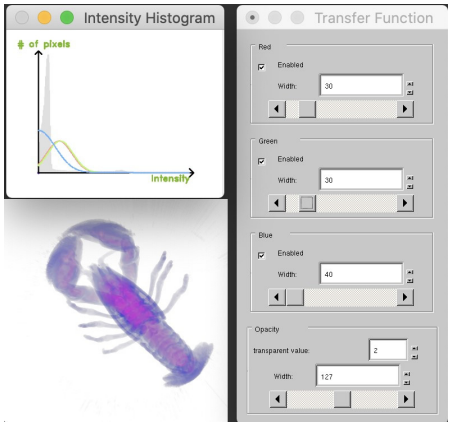

Fig. 5. **Emphasize both the internel part and the external part of the lobster.**

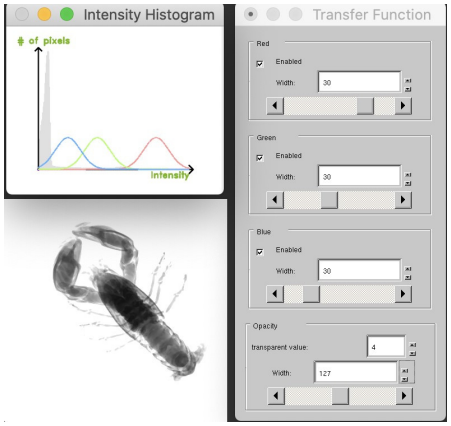

Fig. 6. **Highlight the body and head of lobster using gray scale color only.**

Fig. 6. Lobster

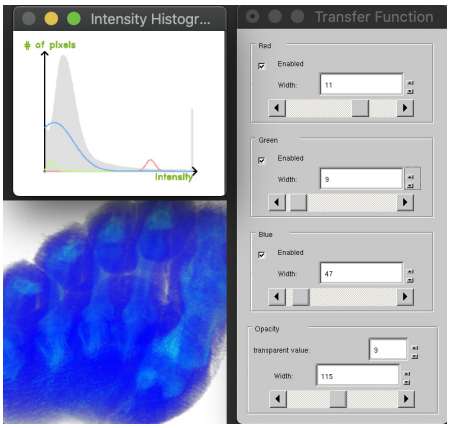

Fig. 8. **Visualize the bones inside the foot by adjusting blue and green color.**

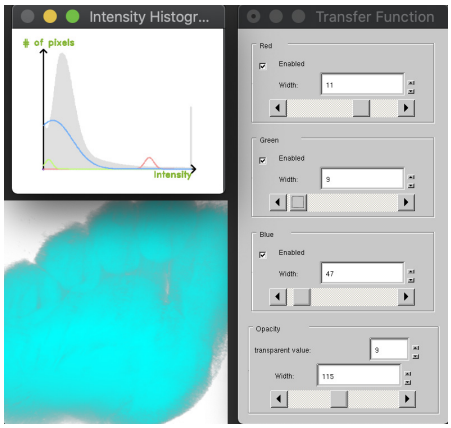

Fig. 9. **By adjusting the width of opacity parameter, we can see the skin of foot.**

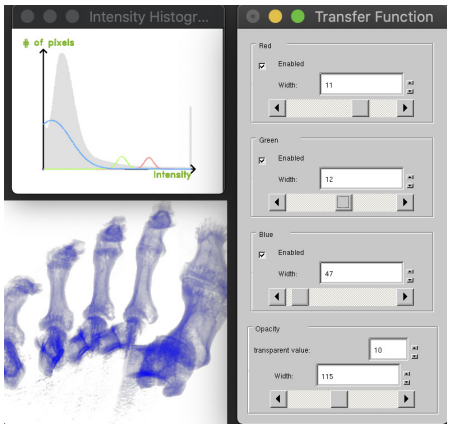

Fig. 10. **By increasing the value of opacity parameter, we can see the bone of foot.**

Fig. 10. Foot

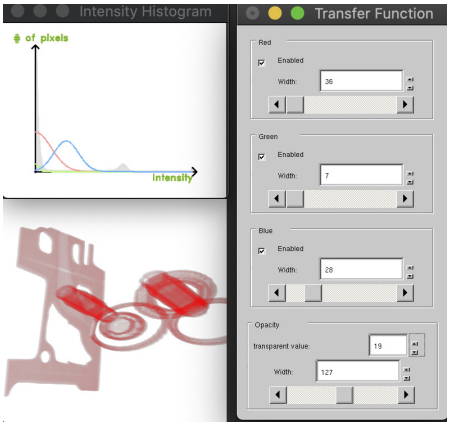

Fig. 12. **By increasing the value of opacity parameter, we can see the inner of the engine.**

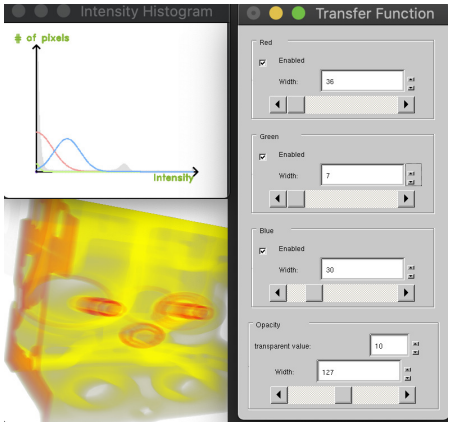

Fig. 13. **By decreasing opacity, we can see both the inner and outer parts with different colors.**

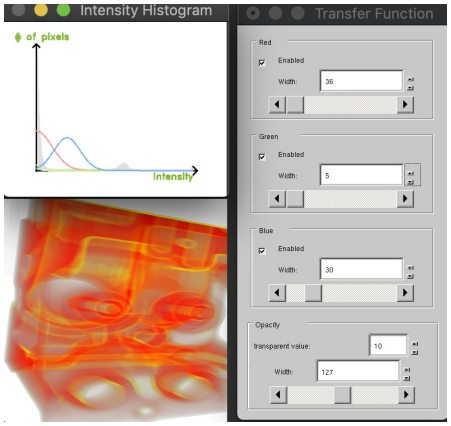

Fig. 14. **When further decreasing opacity, the engine's outer part is visible.**

Fig. 14. Engine

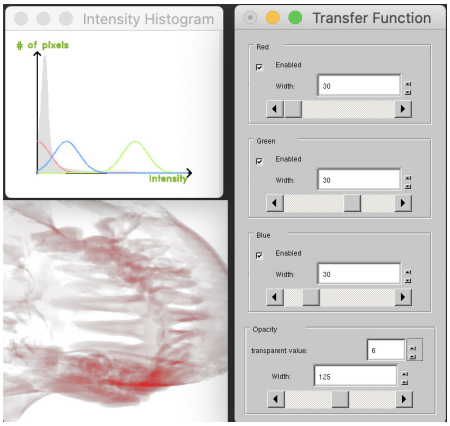

Fig. 16. **We can clearly see the bones of the skull under skin.**

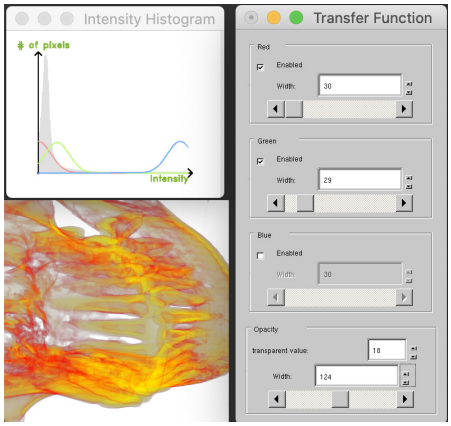

Fig. 17. **The parts where mouth closes has a higher intensity of yellow color.**

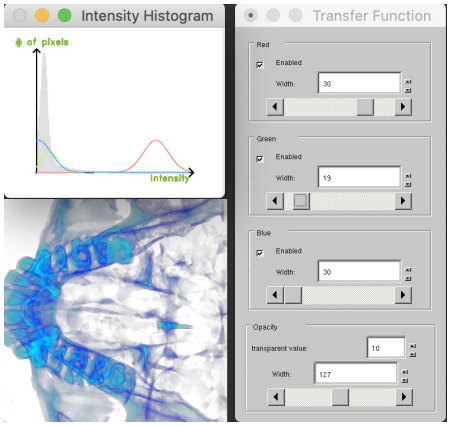

Fig. 18. **By rotating our model by mouse, the teeth can be counted without effort.**

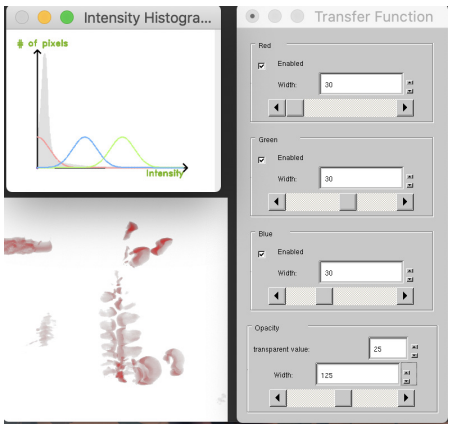

Fig. 19. **We are able to change the settings to render small teeth only.**

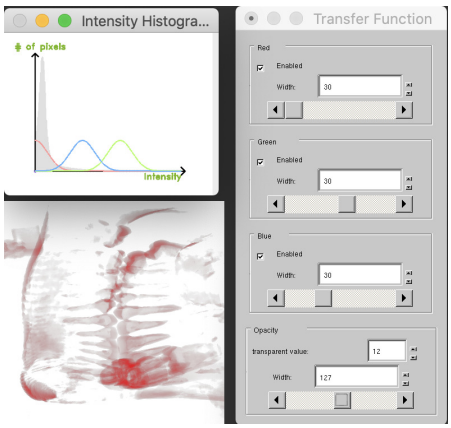

Fig. 20. **Decrease the opacity value and take a different view.**

Fig. 20. Skull

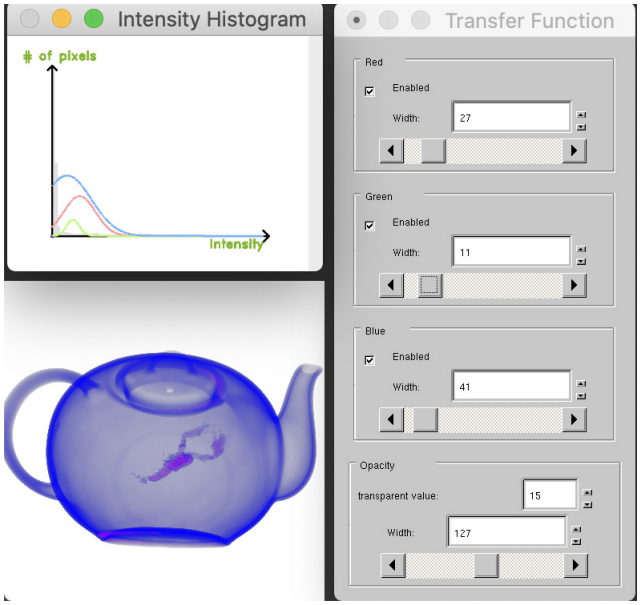

Fig. 22. **The red lobster inside the blue teapot.**

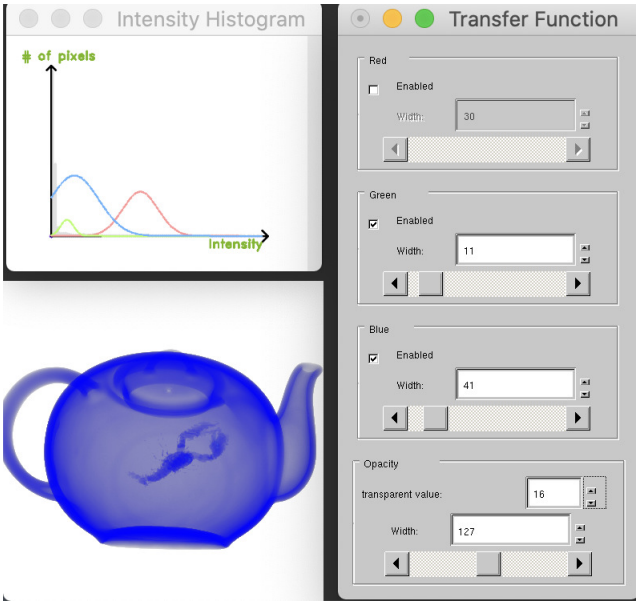

Fig. 23. **Decrease opacity, and the lobster becomes blue.**

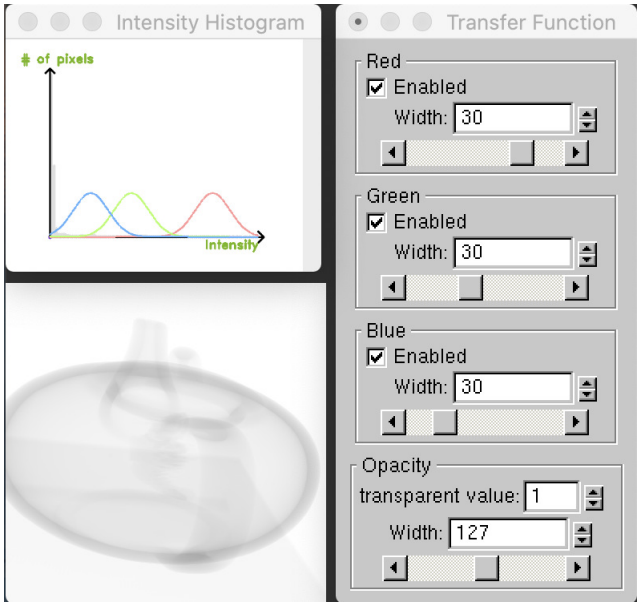

Fig. 24. **The original rendering result without designing transfer function.**

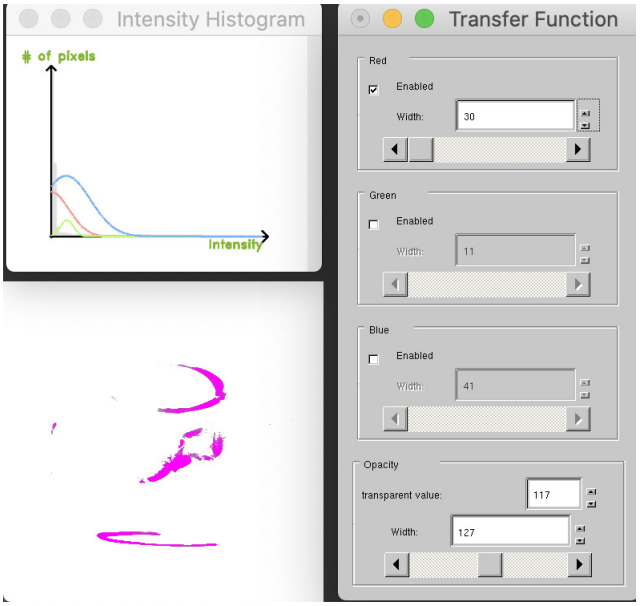

Fig. 25. **Increase the value of opacity, and we can get lobster out.**

Fig. 25. Boston Teapot