Volume Rendering

Aniket G Ninawe

December 2020

1 Introduction

In this assignment, a volume renderer based on the ray-casting approach has been developed. Different types of volume rendering techniques such as have been implemented within the given skeleton code (in Java programming language). The techniques used in the assignment have been explained in the corresponding sections of the report, and illustrated with a few examples built for a number of datasets. Also, the pros and cons of various volume rendering techniques are discussed and several visualizations are illustrated as part of data exploration.

2 Tri-Linear Interpolation and Gradient Computation

Tri-linear interpolation is an interpolation technique used to calculate sample values at non-integer locations, which is particularly useful in ray casting. The interpolated value at a given point X, as illustrated in Figure 1, is calculated using Equation 1[1].

$$\begin{aligned} S_X = &(1-\alpha)\cdot(1-\beta)\cdot(1-\gamma)\cdot S_{X_0} + (\alpha)\cdot(1-\beta)\cdot(1-\gamma)\cdot S_{X_1} + \\ &(1-\alpha)\cdot(\beta)\cdot(1-\gamma)\cdot S_{X_2} + (\alpha)\cdot(\beta)\cdot(1-\gamma)\cdot S_{X_3} + \\ &(1-\alpha)\cdot(1-\beta)\cdot(\gamma)\cdot S_{X_4} + (\alpha)\cdot(1-\beta)\cdot(\gamma)\cdot S_{X_5} + \\ &(1-\alpha)\cdot(\beta)\cdot(\gamma)\cdot S_{X_6} + (\alpha)\cdot(\beta)\cdot(\gamma)\cdot S_{X_7} \end{aligned} \tag{1}$$

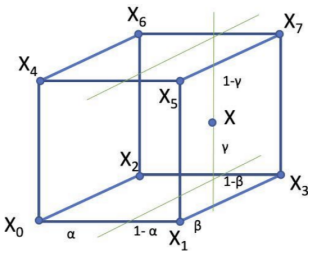

Figure 1: Illustration of the tri-linear interpolation of the point X

Tri-linear interpolation of the gradients is also computed using Equation 1. Each of the X, Y and Z axes is interpolated separately, and the results are subsequently combined together to obtain the final interpolated gradient. Tri-linear interpolation of voxels is implemented in `getVoxelTrilinear` method, and tri-linear interpolation of gradients is implemented in `getGradientTrilinear` method. These methods are available in the class `RaycastRenderer`.

Gradients are computed according to the formula mentioned in the paper by Levoy [2]. Gradient computation takes place in `compute` function in `GradientVolume` class, which is an expensive computation. When the constructor of the class `GradientVolume` is called then only gradient computation takes place.

3 Compositing

Compositing is a technique of aggregation of colors (R,G,B intensities) and opacities of voxels along a ray going through the volume. As opposed to Maximum Intensity Projection (MIP) approach, where only the maximum intensity determines the output, compositing involves weighting of voxels' colors on the basis of their opacity. The resultant colour and opacity values are attributed to the pixel of the view plane, which corresponds to the

[1] Adapted from [1] p. 7

ray. Composting was implemented using the back-to-front algorithm[2] (2) within `traceRayComposite` method of `RaycastRenderer` class.

$$\begin{aligned} C_i &= \tau_i \cdot c_i + (1-\tau_i) \cdot C_{i-1} \\ \alpha_i &= \tau_i + (1-\tau_i) \cdot \alpha_{i-1} \end{aligned} \tag{2}$$

where C_i and α_i are the composite color and opacity, respectively, which are accumulated along the ray, in the direction towards the view point, incorporating the color and opacity (c_i, τ_i) of each new voxel encountered.

Figures 2, 3 show how applying the compositing approach (with the default 1D transfer function) allows to improve the visualization of the *orange* dataset as compared with MIP; shape, visual properties of the object's material, as well as different surface irregularities are more clearly visible when compositing is used. Furthermore, compositing allows to assign different colors to parts of the intensity distribution using 1D transfer functions and, thus, highlight different materials, as illustrated in Figure 4. This figure also illustrates the disadvantage of compositing as compared with MIP: designing a custom transfer function is needed for each dataset to obtain a useful visualization.

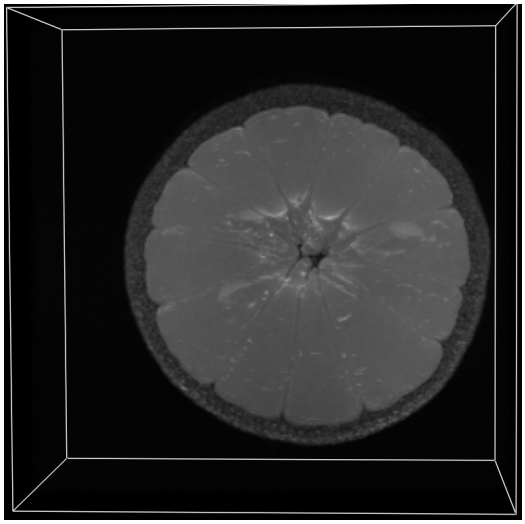

Figure 2: Maximum Intensity Projection (MIP) on *orange* dataset.

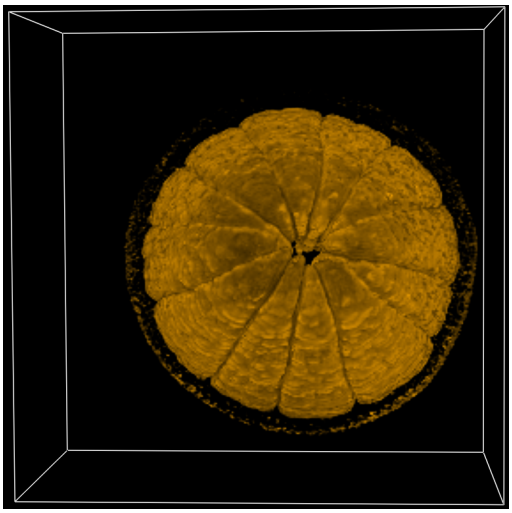

Figure 3: Compositing using the default 1D transfer function on *orange* dataset (without volume shading).

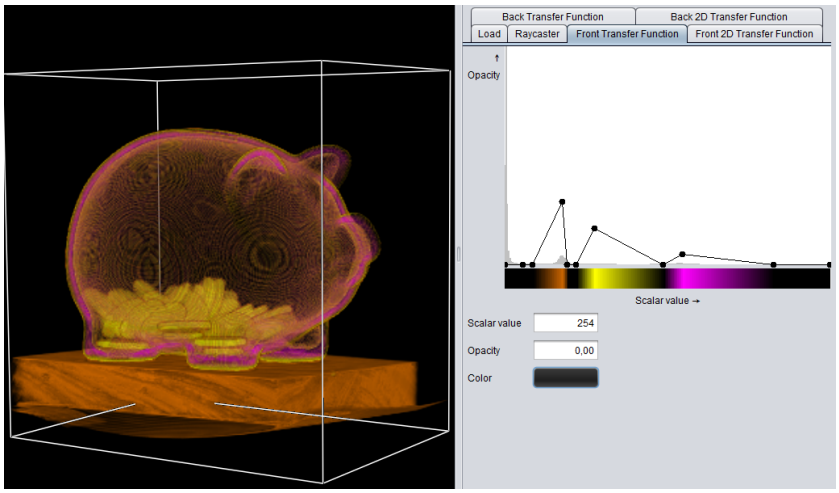

Figure 4: Compositing on *pig* dataset using a custom 1D transfer function (without volume shading).

In order to make the raycaster faster, the resolution of the compositing output during interaction with the volume (`interactiveMode` variable is True) is lowered by increasing the increment step (`increment=2`) and the sample step (`sampleStep=3`); this feature is implemented in `raycast` method.

Further, the application of 2D transfer functions within compositing approach is discussed below in Section 6.

[2] Adapted from [1] p. 31 and https://www.kth.se/social/files/565e35dff27654457fb84363/08_VolumeRendering.pdf

4 Isosurface Rendering

Isosurface rendering is a technique that provides an approximation of the surface, and it was implemented within the method `traceRayIso`. Whether a particular voxel is visible is determined by a threshold on the voxel's value (iso-value). If the value is above the threshold, the voxel is painted with a given color (iso-color). Both iso-value and iso-color are chosen in the UI.

Isosurface rendering helps observe the shape and possibly the surface characteristics of the volume (when a volume shading technique, such as Phong shading (5), is enabled). The higher the isovalue, the fewer pixels are painted and displayed. Such simplicity is the main advantage of the approach. The drawbacks of isosurfacing, according to the lecture slides, are that it only provides an approximation of the surface and is likely to loose information, since no aggregation happens. As a result, this approach is not useful for amorphous bodies.

Figures 5, 6 display the results of applying isosurface rendering. As can be seen, although both pictures have only one color each, surface patterns are visible.

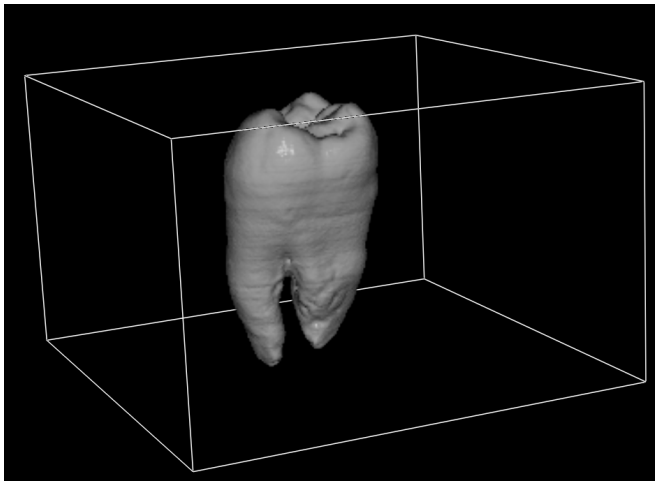

Figure 5: Isosurface rendering on *tooth* dataset (with volume shading).

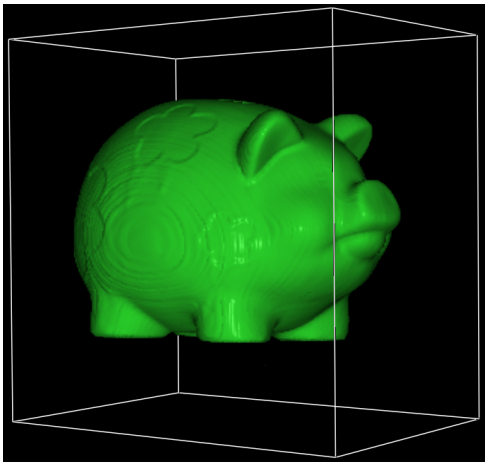

Figure 6: Isosurface rendering on *carp8* dataset (with volume shading).

5 Phong Shading

Phong shading model provides realistic light reflection features to the images. The model has been implemented within `computePhongShading` method according to Equation 3[3]

$$I_i = I_{i,amb} \cdot k_{amb} + I_{i,diff} \cdot k_{diff}(L \cdot N) + I_{i,spec} \cdot k_{spec}(V \cdot R)^{\alpha} \tag{3}$$

where the index i corresponds to the channel (R, G, B) and I_i denotes the channel's intensity of the resulting color. The ambient color $I_{i,amb}$ and the diffuse color $I_{i,diff}$ are defined by the voxel's color, while the specular color $I_{i,spec}$ is assumed to be white. L is the light vector and is assumed to be the same as the view vector V, pointing towards the viewer (headlight); R is the light reflection vector. The parameters are set to the default provided values: $k_{amb} = 0.1$, $k_{diff} = 0.7$, $k_{spec} = 0.2$, $\alpha = 100$.

Figures 8, 10 show the results of applying Phong shading in isosurfacing and compositing modes, respectively, for the *carp8* dataset. The corresponding images without Phong shading are provided on the left in Figures 7, 9. It can be seen that Phong shading technique helps highlight features of the object's surface. It seems essential for the isosurfacing approach, but it also benefits the compositing approach by highlighting the skeleton structure and surface properties.

[3] Adapted from [1] p. 22 and https://www.wikiwand.com/en/Phong_reflection_model

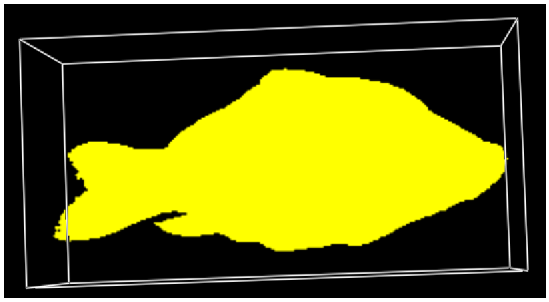

Figure 7: Isosurface raycasting on *carp8* dataset (without volume shading).

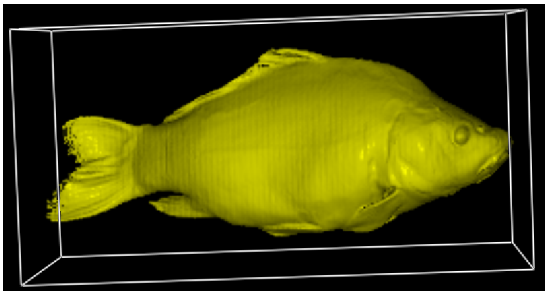

Figure 8: Isosurface raycasting on *carp8* dataset (with volume shading).

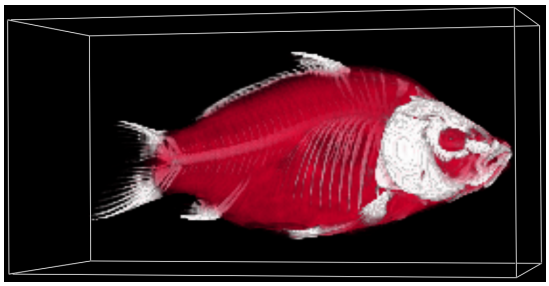

Figure 9: Compositing on *carp8* dataset (without volume shading).

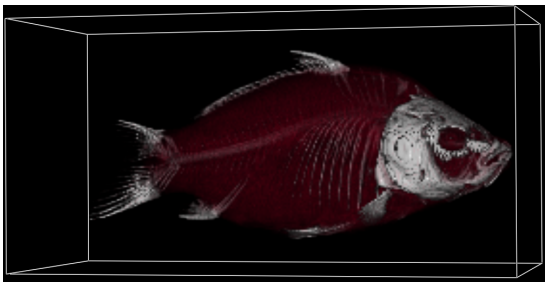

Figure 10: Compositing on *carp8* dataset (with volume shading).

6 2D Transfer Function

Just like coloring a 3D model using a 1D transfer function, the model can also be colored using a 2D transfer function. The pre-implemented 2D transfer function widget is used to adjust the *intensity* and *gradient magnitude*. By properly adjusting the *intensity* and *gradient magnitude*, desired visualization of the model can be obtained. The weight-based opacity is computed as described in [2].

One of the main advantages of using weight based opacity is that the voxels which are present on the edges are highlighted more than the voxels which are present in the center of regions. This allows to visualize the internal parts of a model. This is illustrated in Figures 11 and 12. In Figure 11, we can see the internal organs of the *carp*. In Figure 12, we can see the coins present inside the piggy bank. Even though, the internal details of the model can also be visualized using Compositing (see Figure 4), it requires very detailed modifications in the transfer function to obtain such visualizations, whereas in weight based opacity, such visualizations can be easily obtained very easily with the help of 2D transfer function. Volume shading can be also applied to models along with compositing using a 2D transfer function. This is illustrated in Figure 13. This is implemented in `computeOpacity2DTF`.

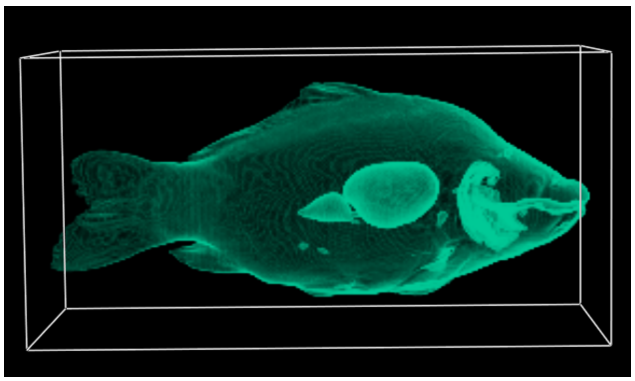

Figure 11: Compositing using 2D transfer function (without volume shading).

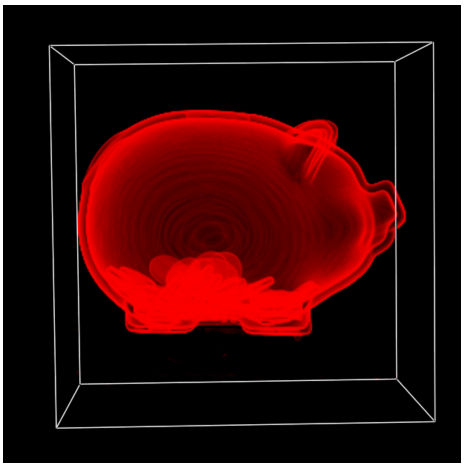

Figure 12: Compositing using 2D transfer function (with volume shading).

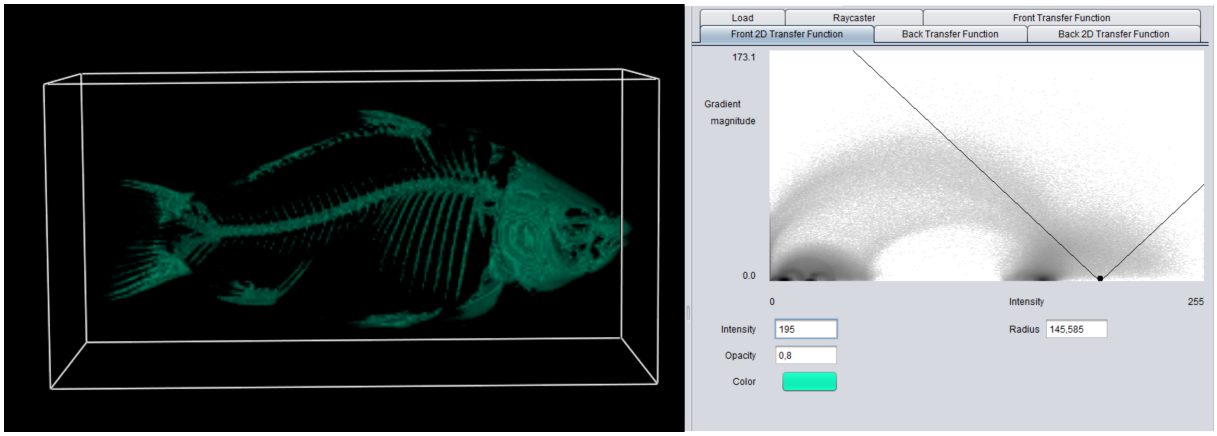

Figure 13: Applying 2D transfer function to *carp8* dataset with volume shading

7 Cutting Plane

A cutting plane allows to apply different visualization algorithms to the half-spaces, into which the plane, controlled by the user, divides the volume. In order to implement a cutting plane within the raycasting renderer, each pixel of the view plane with an associated ray needs to be attributed to one of the half-spaces at any given point in time. This can be achieved by observing the location of the first intersection point (entry point) between the view vector and the volume with respect to the cutting plane. The idea of implementation is shown in Figure 14:

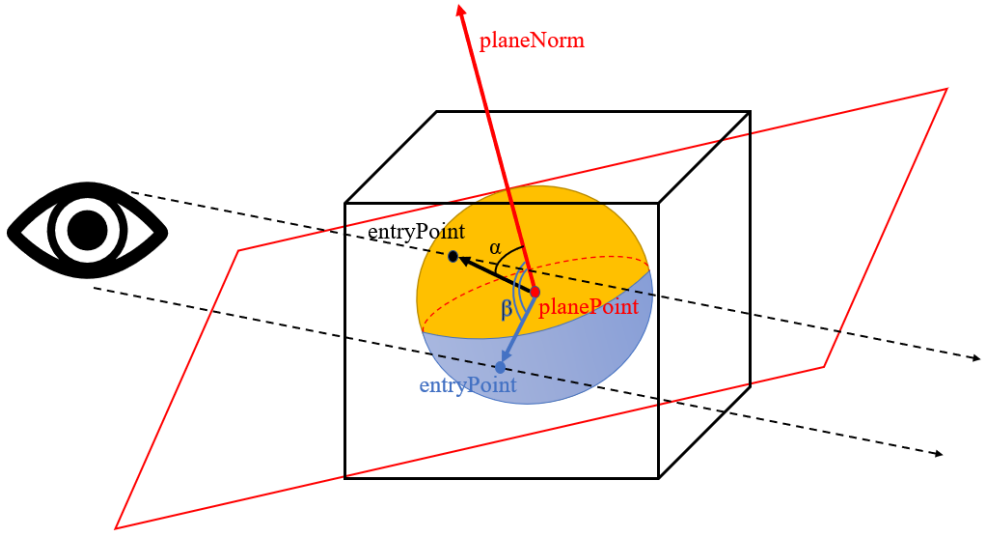

Figure 14: Cutting plane implementation idea.

As can be seen, the cutting plane is represented in point-normal form, that is, given a point lying on the plane and its normal vector. Calculating the dot product between the normal vector and a vector induced by an entry point and the plane point suffices to determine to which side a particular view vector should be attributed. A non-negative dot product suggests an angle $0° \leq \alpha \leq 90°$ and, therefore, the front half-space. Conversely, a negative dot product suggests an obtuse angle $90° < \beta < 180°$and the back half-space. The cutting plane was implemented inside the loop in `raycast` method; the methods corresponding to different raycasting techniques (MIP, Compositing, Isosurfacing) were modified to include a boolean variable, as defined in (4), based on which essential variables are determined for a particular ray: 1D and 2D transfer functions, iso-values and iso-colors.

$$frontPlaneBoolean = \begin{cases} True, & \text{if } (entyPoint - planePoint) \cdot planeNorm \geq 0 \\ False, & \text{if } (entyPoint - planePoint) \cdot planeNorm < 0 \end{cases} \tag{4}$$

Using a cutting plane allows to obtain useful insights as a result of combining features of different visualization techniques. For, example, using one technique to capture the surface features and another to look inside the volume. Two examples of cutting plane application are shown in Figures 15, 16, 17 and 18.

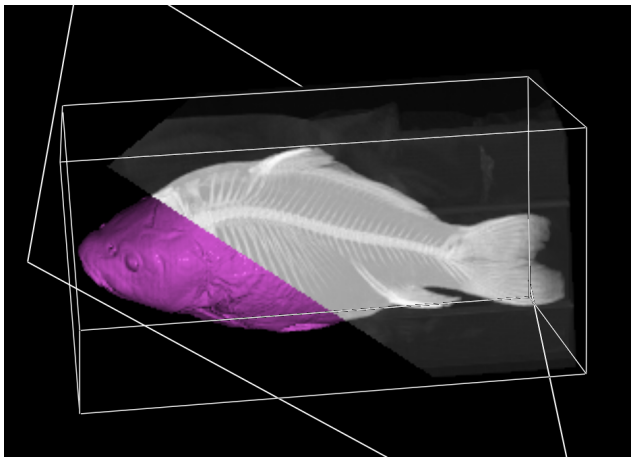

Figure 15: Using a cutting plane to combine MIP and Isosurfacing on *carp8* dataset (Phong shading enabled).

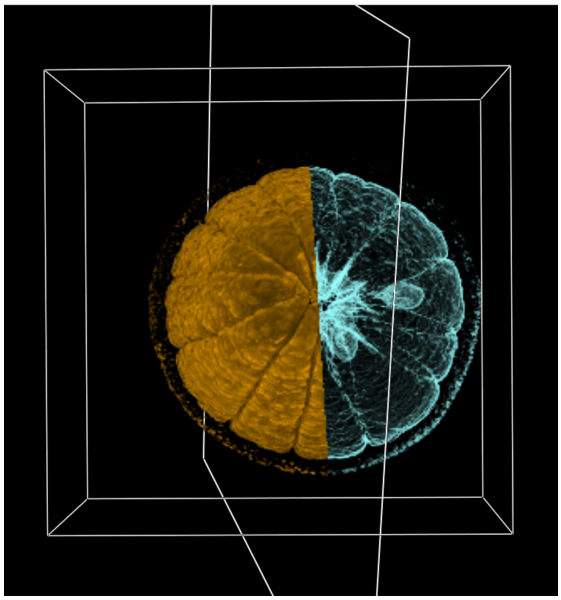

Figure 16: Using a cutting plane to illustrate compositing with 1D and 2D transfer functions simultaneously on *orange* dataset.

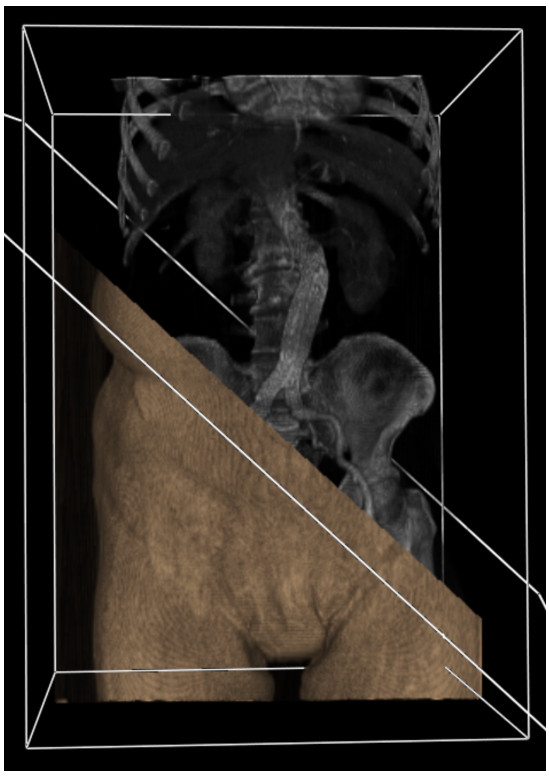

Figure 17: Using a cutting plane to illustrate compositing with 1D and 2D transfer functions simultaneously on *stent* dataset.

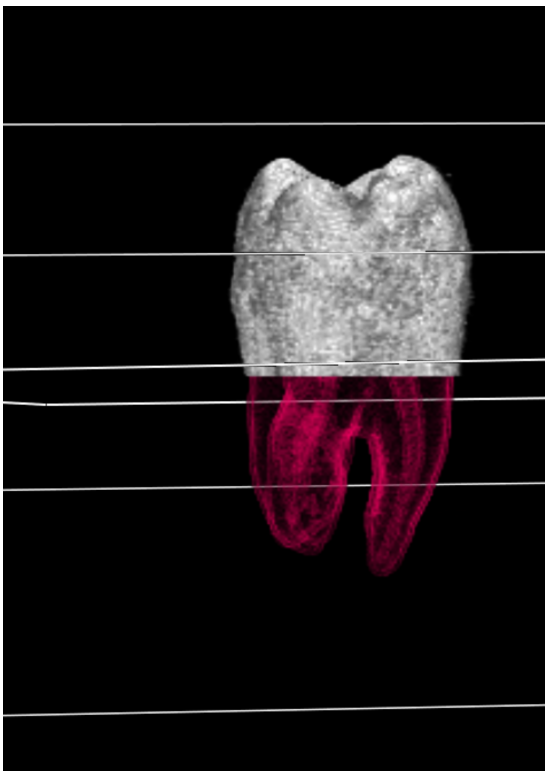

Figure 18: Using a cutting plane to illustrate compositing with 1D and 2D transfer functions simultaneously on *tooth* dataset.

8 Conclusion

In the present report, the results of raycaster implementation and the findings observed were presented. Different methods of volume rendering were discussed: MIP, isosurface rendering, and compositing with 1D and 2D transfer functions. Isosurface rendering was used to observe the shape and surface features of the objects. To improve the result, making the shape more clear and highlighted, the isosurface rendering can be combined with Phong shading. Compositing was used to apply an opacity-based weighting algorithm to the colors of voxels along a ray and, thus,

aggregate information. Compositing was shown to be capable of providing useful visualizations when used along with either a 1D or a 2D transfer function.

A number of datasets were explored with these rendering techniques. The results, explorations and observations have been discussed in each section above. The attached figures in different sections demonstrated the results of applying the discussed techniques.

9 Individual Contribution

All the team members were equally involved during various stages of the assignment. All team members have equal contribution in the implementation and report.

References

[1] 2IMV20 Visualization course slides, 2020. Encoding: spatial data (week 2).

[2] Marc Levoy. Display of surfaces from volume data. *IEEE Computer graphics and Applications*, 8(3):29–37, 1988.

Appendices

A Figures: Exploration of different datasets

In this section, several additional visualizations of the datasets are provided as a result of applying different rendering techniques, as shown in Figures 19, 20, 21, 22, 23. .

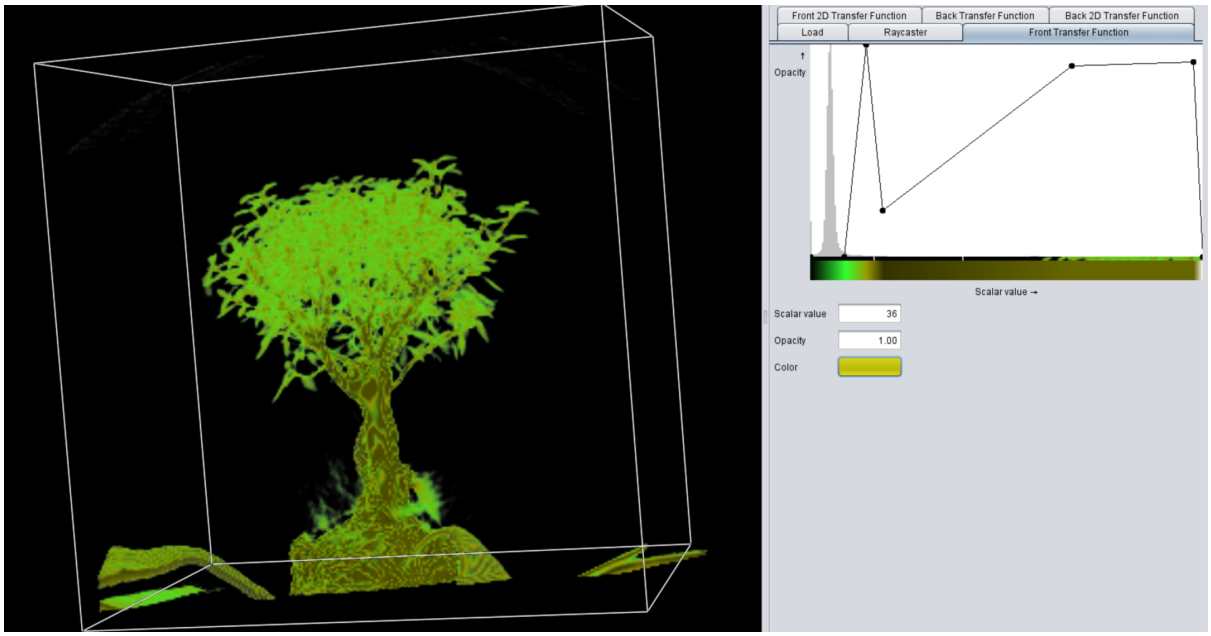

Figure 19: Compositing on *bonsai* dataset (without Volume Shading)

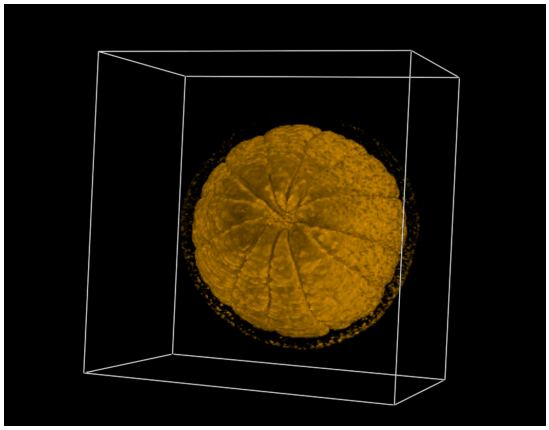

Figure 20: Compositing on Orange dataset without Volume Shading.

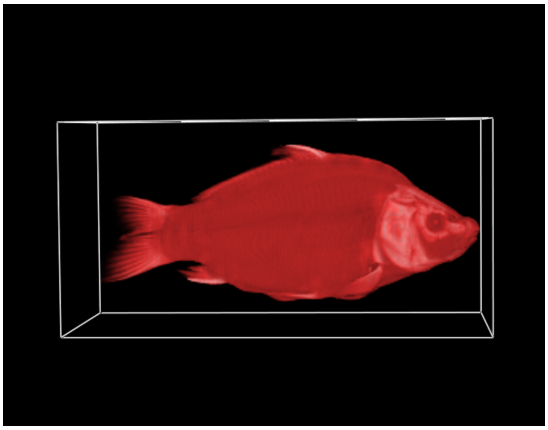

Figure 21: Compositing on Carp dataset without Volume Shading.

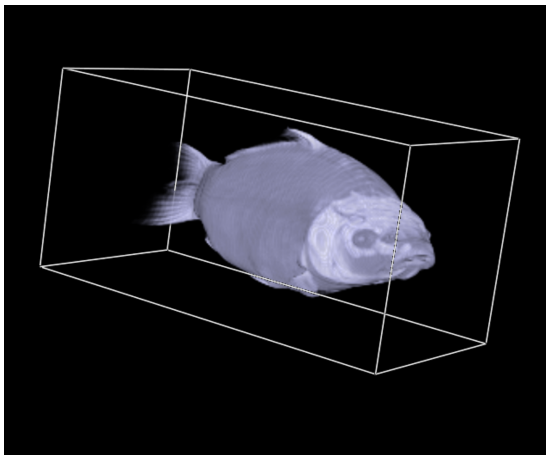

Figure 22: Compositing on Carp dataset without Volume Shading rotated.

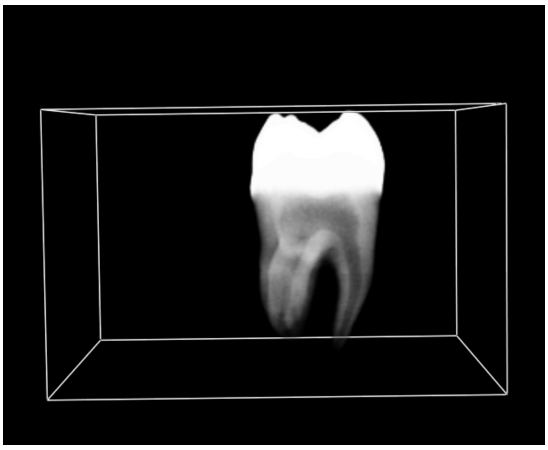

Figure 23: Compositing on Tooth dataset without Volume Shading.