IN4086 Volume Visualization Report Group 13

Wenyu Gu	Tian Tian	Zheng Liu
4896246	4818776	4798406

February 4, 2019

1 Part1: Ray casting

For this part, we implemented following functions:

- tri-cubic interpolation
- additive compositing ray function
- improvement of responsiveness

1.1 Tri-cubic Interpolation

1.1.1 Idea

We implemented *getVoxelTriCubicInterpolate()* in *Volume* class, which takes the coordinates of a point as input and returns its interpolated value. This interpolation method used 64 samples to interpolate a value. Sometimes the value exceeded the original data range, which in this case was 0 to 255, so a check was executed in the end. Result of a slice of a carp is shown in Figure 1.

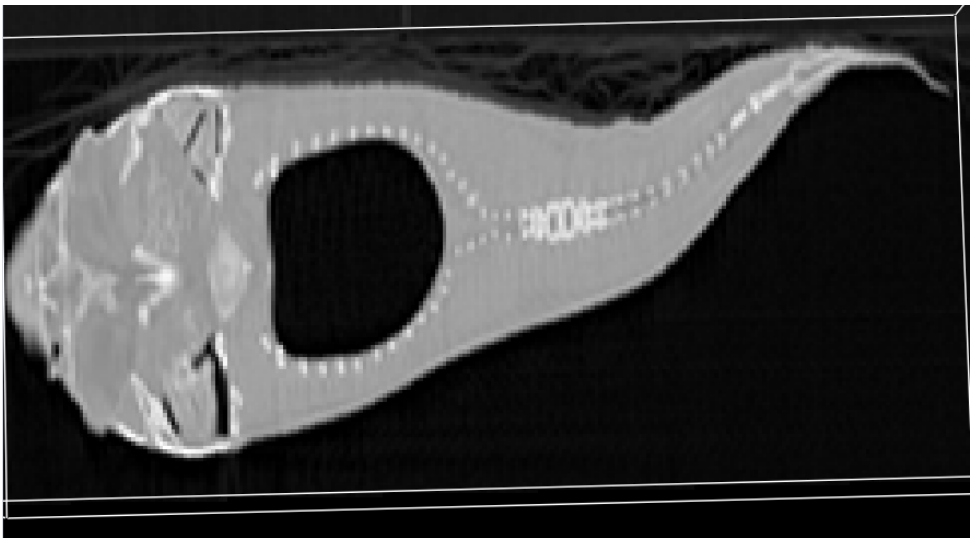
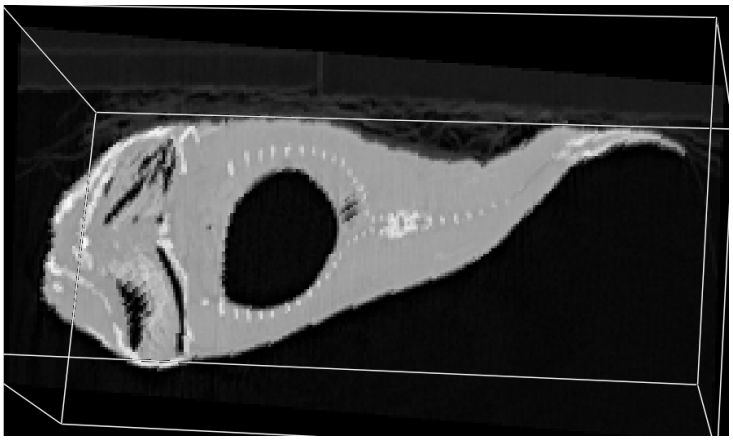


Figure 1: Slicer of a carp using tri-cubic interpolation at resolution 1

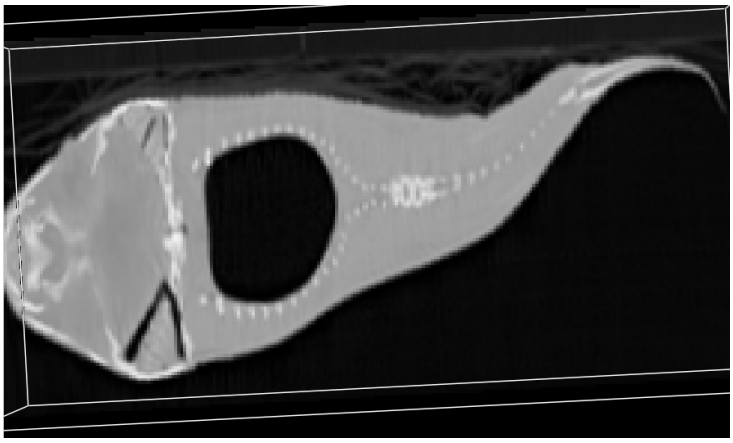
1.1.2 Results

We compared the effect of slicer by using three different interpolation methods: nearest neighbor, linear interpolation and cubic interpolation. Results are shown in 2a and 2b. **Nearest neighbor** interpolation took minimum computation time among three. It took around 0.57 milliseconds to get a full slicer of a fish by using nearest neighbor interpolation. The boundary of fish was quite rough and uneven. **Linear** interpolation used around 0.63 milliseconds to resample an image and the boundary was smoother compared with the previous image. **Cubic** interpolation

took the longest time, 32.32 milliseconds, and it achieved a really smooth border if we zoomed in. But the question is sometimes it might generate negative value out of original data range.



(a) in nearest neighbor interpolation at resolution 1



(b) in linear interpolation at resolution 1

Figure 2: Slicer of a carp

1.2 Compositing Ray Function

1.2.1 Idea

The additive compositing functionality was achieved through *colorCompositing()* in *RaycastRender* class. We computed samples along the ray from the exit point to the entry point and used Equation 1 to calculate compositing color at each point. The color before compositing at each point was obtained from default 1D transfer function provided in skeleton code. In the equation, C_i and A_i represents the compositing color and opacity at each point. We used back-to-front to conduct compositing instead of front-to-back in order to avoid recursion.

$$\begin{aligned} C_i &= c_i * a_i + (1 - a_i) * C_{i-1} \\ A_i &= a_i + (1 - a_i) * A_{i-1} \end{aligned} \tag{1}$$

1.2.2 Results

We examined the result on carp dataset first, which is shown in Figure 3. In this visualization, the skeleton of a carp is white and its skip is transparent. This is because samples belonging to the skeleton carry higher values than samples from skin.

There are two kinds of compositing ray function: additive and maximum. To compare these two approaches, We examined maximum intensity projection(MIP) on the same carp dataset in the following.

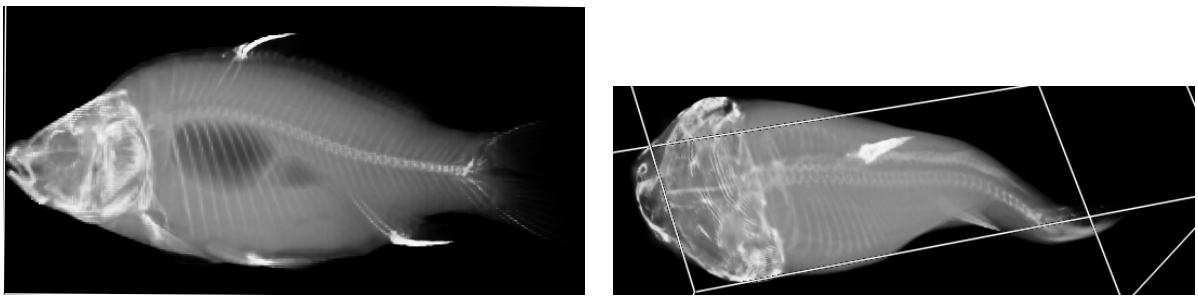


Figure 3: Visualization of a carp using additive raycasting

In Figure 4 the visualization of carp by MIP is shown. The carp looks more "white" as its skin is also painted in white. This is because MIP uses a sample, which possesses the largest value along the way, to compute volume projection. Opacity is not accumulated as in additive projection, making it a bit difficult to see the inside pf carp. **Additive** raycasting(with default 1D black-white transfer function) performed better than **MIP** in highlighting skeleton area.

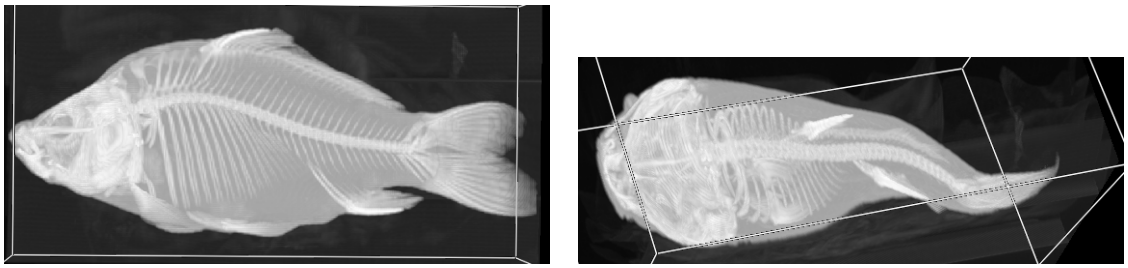
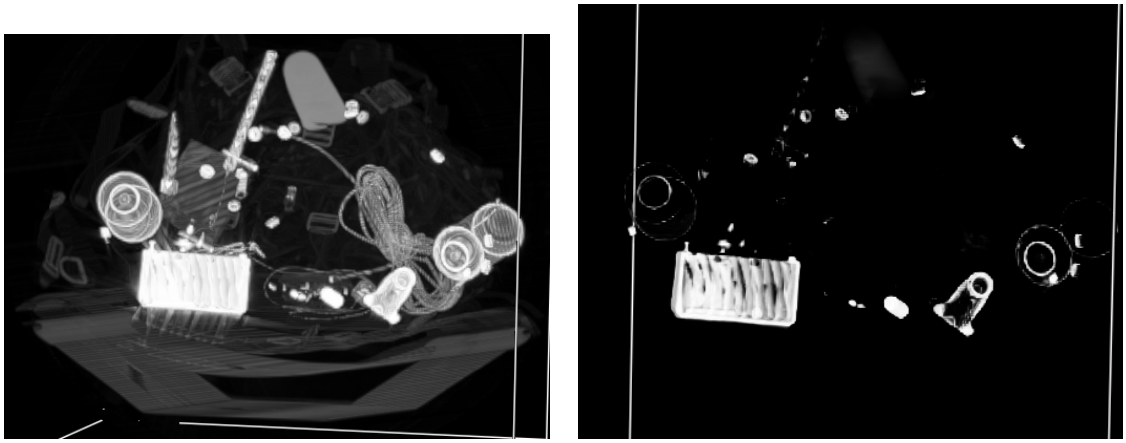


Figure 4: Visualization of a carp using maxmium raycasting

We also tested performances of two raycasting approaches on backpack and pig dataset. As shown in Figure 5, the backpack in additive raycasting only contains part of items, such as rope. The area left is nearly in black, losing a bunch of detail. On the contrary, MIP clearly shows the stuff inside the backpack.

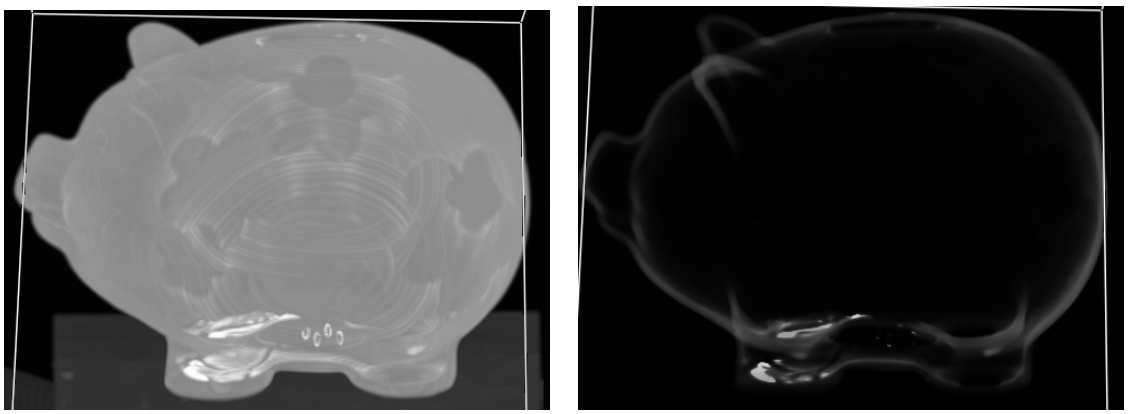


(a) A backpack in mip (b) A backpack in additive raycasting

Figure 5: Visualization on backpack dataset

The result on pig dataset is also similar. In Figure 6, we could only see a rough curve of pig in **additive** compositing mode. The most voxels in the figure is in black. However, in **MIP** the surface of pig is depicted clearly as well as its knuckle.

It could be concluded that when there is complicated structure inside the object, or in the perspective of data when sample values vary significantly, **additive** compositing performs better. otherwise **MIP** might be a better choice.



(a) A pig in mip (b) A pig in additive raycasting

Figure 6: Visualization on pig dataset

1.3 Responsiveness improvement

In order to make the application more responsive during user interaction, we decided to lower the resolution in *interactiveMode*. A button was added to control the mode of computation. In interactiveMode, resolution is set to be level 1. In this way we decreased number of voxel to be computed and improved responsiveness.

2 Part2: Isosurface Raycasting and Shading

For the second part, we implemented the following functions:

- Isosurface Raycasting
- Binary Search of Isosurface
- Tri-linear Interpolation of Gradient
- Phong Shading Model
- Extension: Tone Shading Model

2.1 Isosurface Raycasting

2.1.1 Idea

We implemented *traceRayIso()* in *RaycastRenderer* class to access the isosurface of the object. This is achieved by traversing data along the ray and searching for a voxel whose value is higher than the isovalue and value of previous voxel is lower than that. If such a voxel exists, the point on the image plane is considered to be inside the boundary and is painted in isocolor, otherwise it is regarded as environment and colored in black.

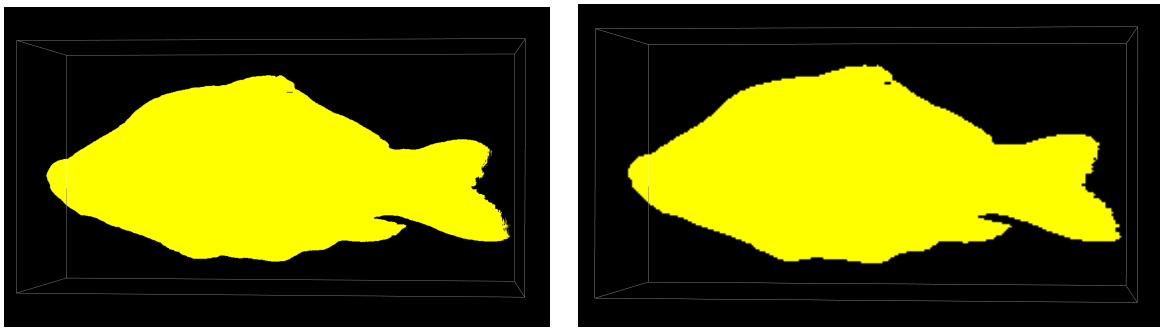
2.1.2 Results

We examined the results on carp dataset, shown in Figure 7, setting the resolution in 4 and 1 respectively, it is obvious that when the resolution is higher, the edge of isosurface is smoother because more details is taken into consideration. However, high resolution results a slow calculating, the time to show result for which in resolution 1 is 122 ms, but approximately 1259 ms for when the resolution is 4.

2.2 Binary Search

2.2.1 Idea

We used *bisection()* in *RaycastRender* to find a more accurate position based on result from isosurface. The idea is from isosurface searching we infer the boundary at first sample value



(a) Isosurface of a carp with isovalue 95 at resolution 4

(b) Isosurface of a carp with isovalue 95 at resolution 1

Figure 7: Visualization of isosurface on dataset carp8

whose value exceeds isovalue. However the accurate position of boundary locates between previous sample and current sample. So we recursively computed the value at middle of the array and compared it with isovalue. The error was set to be 1e-1. To handle the stackoverflow error a try catch clause was used.

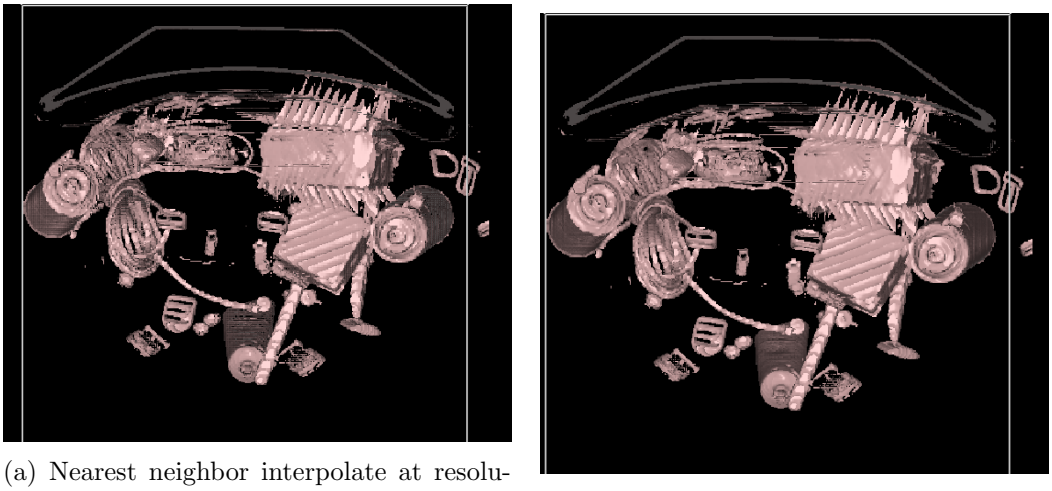
2.3 Tri-linear Interpolation of Gradient

2.3.1 Idea

Similar with idea in Section 1.1, linear interpolation of gradient uses gradients from eight nearby samples to estimate at a specified location. This function is done in *getGradient()* in *GradientVolume* class.

2.3.2 Results

We compared the effect using different gradient interpolation methods, which is shown in Figure 8. By nearest neighbor interpolation we could see there are some noise on the boundary. While in linear interpolation this situation improved a lot.



(a) Nearest neighbor interpolate at resolution 2

(b) Linear interpolate at resolution 2

Figure 8: Different gradient interpolation on Phong shading model

2.4 Phong Shading

2.4.1 Idea

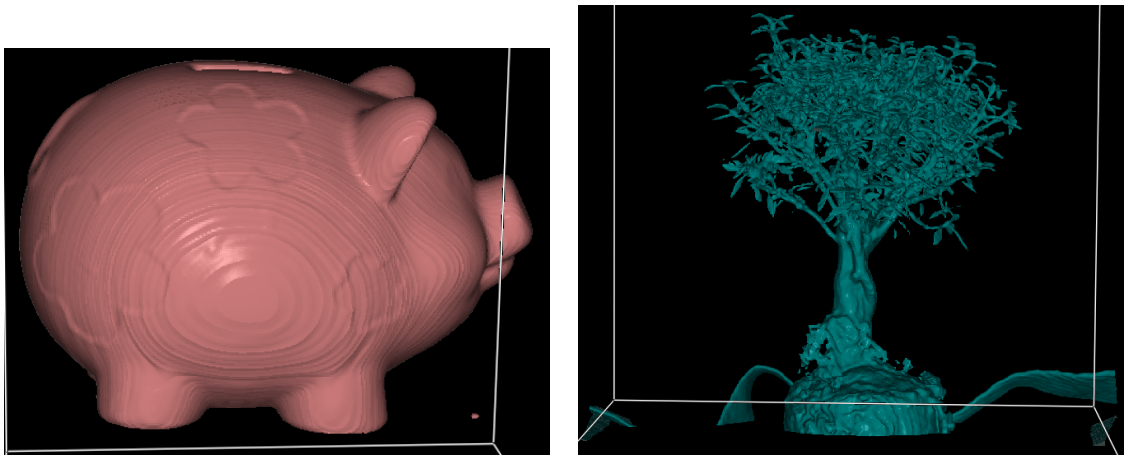
Phong shading [2] was used to add shading on 3D structure to 2D plane and was implemented in *computePhongShading* in *RaycastRender* class. We computed Phong shading based on following Equation:

$$\begin{aligned} C_{phong} &= C_{ambient} + C_{diffuse} + C_{specular} \\ C_{ambient} &= k_a * C_a * L_a \\ C_{diffuse} &= k_d * C_d * L_d * \cos\theta \\ C_{specular} &= k_s * C_s * L_s * (\cos\phi)^n \end{aligned} \tag{2}$$

where L_a, L_d, L_s stands for light property color, K_a, K_d, K_s represents material constants, C_a, C_d and C_s is the material color, and C_s is usually white.

2.4.2 Results

We tested the performance of Phong Shading combined with different raycasting method. Combined with isosurface, Phong shading helps add light and shade on image and makes it look more like a 3D object, instead of using a same predefined color. In Figure 9 shows two examples of Phong shading with isosurface. Figure 10 shows a break down of Phong shading on carp dataset.



(a) Pig (b) Bonsai

Figure 9: Phong shading with isosurface raycasting

Combined with compositing, the effect of Phong shading is shown in Figure 11.

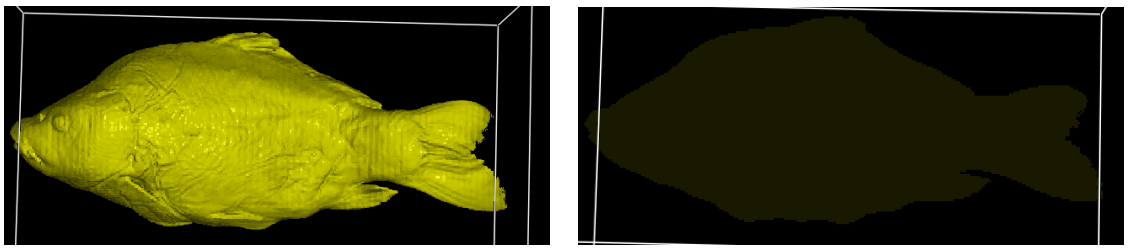
2.5 Extension: Tone Shading

2.5.1 Idea

Tone shading was used to extent and explore more shading method and its related effects. The basic idea is that tones are considered a crucial concept to illustrators [1, 3]. We mixed cold and warm color, also known as different temperature of colors, they vary in relative contribution across the surface. The following equations offer the methods to calculate tone shading model.

$$Color_{cold} = Color_{cold\_original} + \alpha \times Color' \tag{3}$$

$$Color_{warm} = Color_{warm\_original} + \beta \times Color' \tag{4}$$



(a) Carp in Phong shading

(b) Carp in ambient light

(c) Carp in diffuse light

(d) Carp in specular light

Figure 10: Decompositing Phong shading

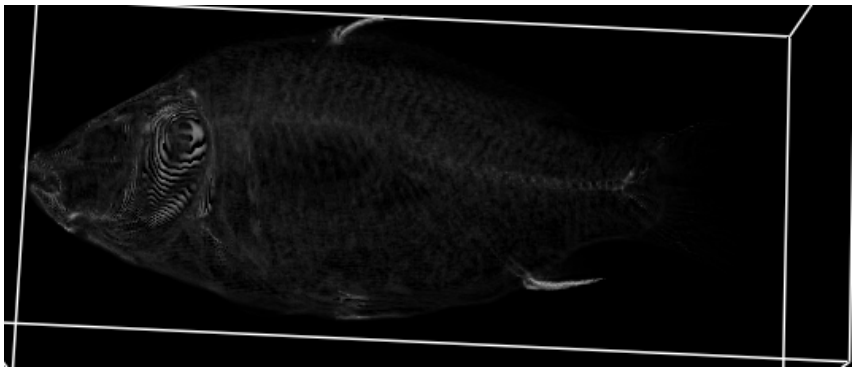


Figure 11: Phong shading with compositing raycasting

$$I = \left(\frac{1+\widehat{l} * \widehat{n}}{2}\right) k_{cold} + \left(1 - \frac{1+\widehat{l} * \widehat{n}}{2}\right) k_{warm} \tag{5}$$

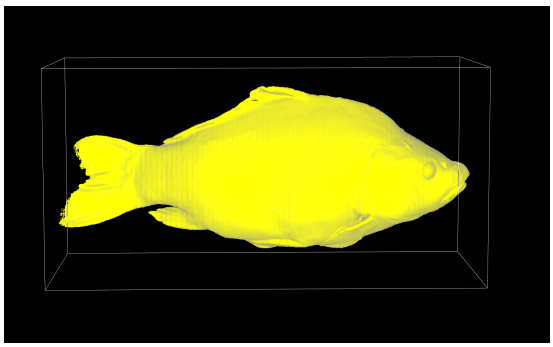
2.5.2 Result

We tested the performance of Tone shading model and combined with isosurface, compositing and 2D transfer function respectively. The visualisation results are shown in figure 12. It illustrates a clear effect on combination with compositing, but performs not well in both isosurface and 2D transfer function. For isosurface, tone shading does not illustrate the detailed structure of crap and for 2D transfer function, it concludes to many noises.

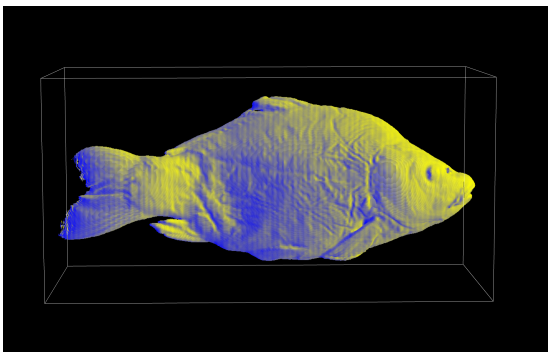
3 Part3: Transfer Function

For the third part of project, we achieved following function:

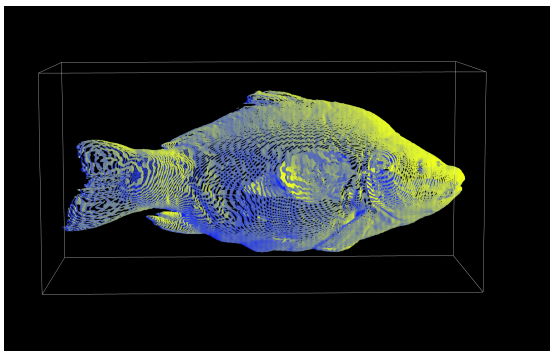
- 2D Transfer Function



(a) Tone shading with isosurface at resolution 4



(b) Tone shading with compositing at resolution 1



(c) Tone shading with 2D transfer function at resolution 4

Figure 12: Visualization of tone shading on dataset carp8

3.1 1D Transfer Function Exploration

3.1.1 Results & Evaluate

Figure 13 shows the result of compositing by using 1D Transfer Function on the pig dataset. Comparing with **MIP** in Figure 6, compositing method shows more flexibility than MIP. We can easily observe the flower on the surface and the coins inside by using this method. That is to say, it could show different levels of the object, while MIP could not. However, it still has its disadvantages like the overlap between two groups could not be visualized separately. This problem is solved by 2D Transfer Function.

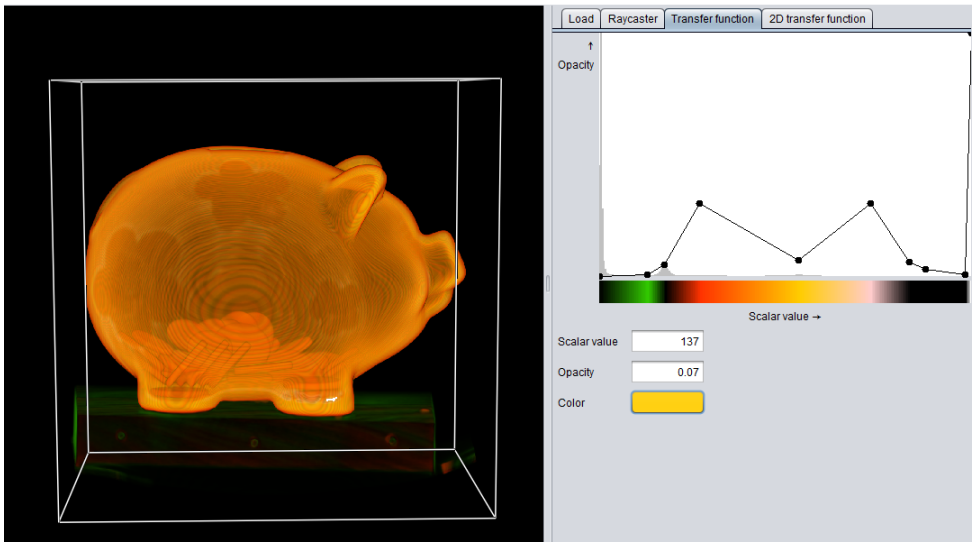


Figure 13: Compositing on pig dataset

3.2 2D Transfer Function

3.2.1 Idea

2D Transfer Function calculates opacity based on the value of voxel and its gradient magnitude, which is implemented in *computeOpacity2DTF()* in *RaycastRenderer* class. It used a triangle widget. Voxels mapped outside the triangle has a zero opacity. Voxels in inner triangle possess a non-negative opacity. These two groups are classified by slope. Inside the triangle from central to border the opacity goes linearly from 1 to 0.

3.2.2 Results

We examined the results on carp dataset. Figure 14 and figure 15 showed the results by using 2D Transfer Function with different parameters settings. In addition, we combined the result by using 2D Transfer Function and volume shading in figure 16. Comparing with the 1D Transfer Function, we found that this method could show more information on the flower pattern. The result in figure 13 only shows the flower pattern, but not clear whether the patterns were stickers, concaves or convexes. In figure 16, we could see the pattern is concave.

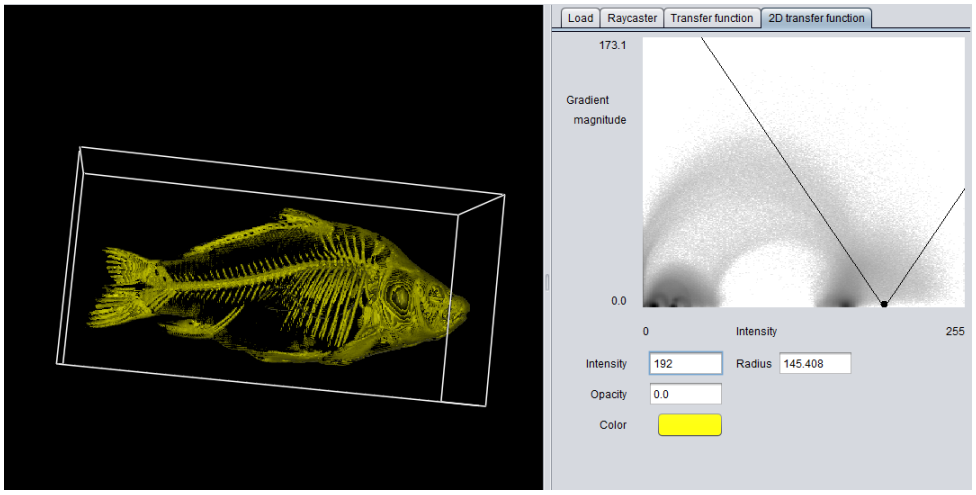


Figure 14: 2DTF results on carp dataset with setting 1

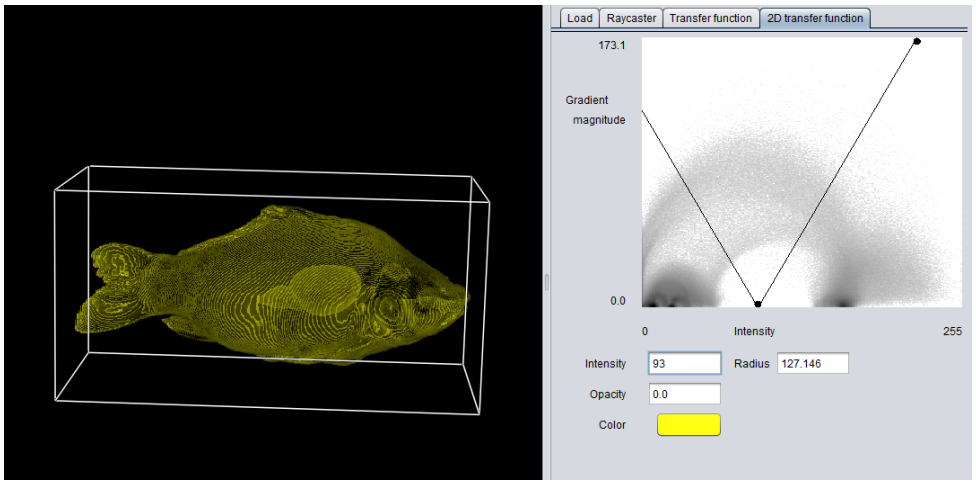


Figure 15: 2DTF results on carp dataset with setting 2

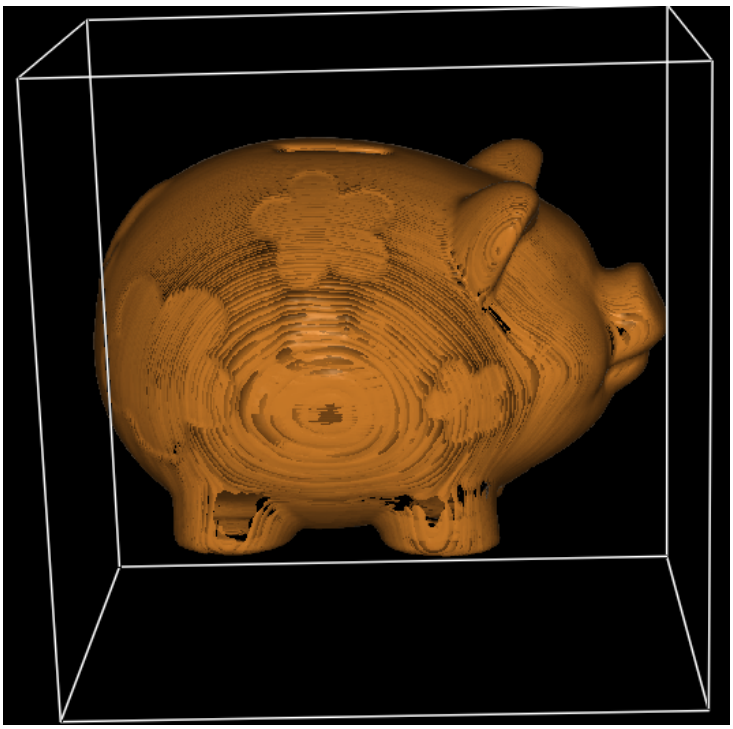


Figure 16: 2DTF results on pig dataset

References

[1] Bruce or Amy Gooch. Tone-based shading of matte objects. https://www.cs.northwestern.edu/ ago820/SIG98/paper/node6.html. Accessed April 21, 1998.

[2] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[3] Penny Rheingans and David Ebert. Volume illustration: Nonphotorealistic rendering of volume models. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):253–264, 2001.

A Individual Report

A.1 Wenyu Gu

I got to know visualization through this project. For me it is interesting to see how code changes could have an effect on final visualization, so I had a lot initiative to invest in this assignment. My major contribution was code for part one, three and Phong shading in part two. I also wrote report for part one. I started a little bit early than my teammates thanks for not-that-busy post-Chirstmas week. Before coding, I discussed the theoretical knowledge with my teammates, and we all reached an agreement. My teammates also helped me a lot with the code and bugs.

A.2 Zheng Liu

I applied the volume visualisation knowledge by doing this project. I mainly focus on the part two and implemented the extension of Tone shading mode. I wrote the report for the related part. I and my teammates solved most of the theoretical problems together and finally applied them into the visualisation application. We do the project based on the evaluation sheet step by step, which we did efficiently. It is my first time to write Java code, visualisation knowledge is not the only gain, now I can read Java code fluently and can apply them into project.

A.3 Tian Tian

I learned a lot through this volume visualization project. In this project, I take in charge of Part III: 2D transfer function. At the beginning, I have no idea what to start with and how to write the code. My teammates helped me a lot on code writing since I have been a long time not writing code in Java. In addition, I also wrote the report for part three. After discussing with my teammates, we come up with a best way to deliver our design through the report. We evaluated our results in different datasets and discussed the different methods to display the results. We found that each method had its pros and cons. In conclusion, I had a enjoyable time in this project with my teammates. We cooperated and completed this project together.