Visualization

Assignment 1

Irfan Nur Afif (1035476)
Miranti Intan Rahmani (0999432)

December 12, 2016

Contents

1 Ray Casting **3**
1.1 Implementation . . . 3
1.1.1 Trilinear Interpolation . . . 3
1.1.2 Maximum Intensity Projection . . . 4
1.1.3 Compositing Ray Function . . . 4
1.1.4 Responsiveness Improvement . . . 5
1.2 Data Exploration . . . 5
1.2.1 Maximum Intensity Projection . . . 5
1.2.2 Compositing Ray Function . . . 5
1.2.3 Responsiveness . . . 6
1.3 Result . . . 7

2 2D Transfer Function **7**
2.1 Implementation . . . 8
2.1.1 Gradient-Based Opacity Weighting . . . 8
2.1.2 Extended Triangle Widget . . . 9
2.1.3 Illumination Model . . . 9
2.2 Data Exploration . . . 10
2.2.1 Gradient-Based Opacity Weighting . . . 10
2.2.2 Extended Triangle Widget . . . 10
2.2.3 Local Illumination . . . 11
2.3 Result . . . 11

1 Ray Casting

In the first assignment, we are expected to implement volume rendering technique to create visualization of three-dimensional data by extending the skeleton code that has already provided. The technique implemented is based on raycasting approach.

1.1 Implementation

The idea behind Direct Volume Rendering using ray casting method is that we cast a ray from the observer's point of view perpendicular to the view plane, passing through the volume data, take samples of the voxel along the ray, map the sample value into color and opacity using transfer function, and project the output into the image pixel. The scalar values of sample voxels can be used directly to obtain the view (in terms of intensity) using Maximum Intensity Projection, or mapped through the transfer function to obtain the colors and opacities. A gradient function can also be added into the transfer function to gain more features of the datasets.

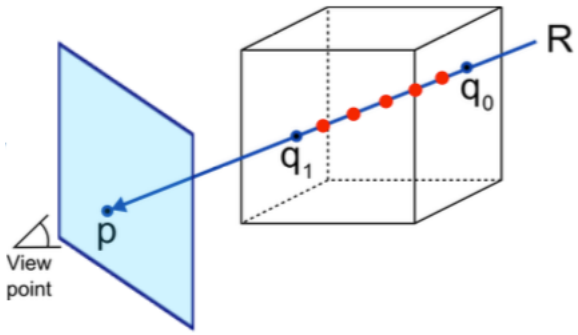


Figure 1: Volume Rendering using Ray Casting method

1.1.1 Trilinear Interpolation

Theoretically, the data point of a voxel can be obtained by calculating the sum of the products of the value at each corner and the partial volume diagonally opposite the corner[1]. The interpolation process in general is visualized in the Figure 2. Mathematically, the data point is calculated as follows:

$$\begin{aligned} S_X = (1-\alpha)(1-\beta)(1-\gamma)S_{X0} + \alpha(1-\beta)(1-\gamma)S_{X1} + \\ (1-\alpha)\beta(1-\gamma)S_{X2} + \alpha\beta(1-\gamma)S_{X3} + \\ (1-\alpha)(1-\beta)\gamma S_{X4} + \alpha(1-\beta)\gamma S_{X5} + \\ (1-\alpha)\beta\gamma S_{X6} + \alpha\beta\gamma S_{X7} \end{aligned}$$

This mathematical formula is implemented in the method *getVoxelByInterpolation()* which extends the original *getVoxel()* method in *RaycastRenderer* class. It takes three dimensional coordinate as an input. This coordinate represents the point where the ray hits in the voxel plane. We define 8 points (from X0 to X7) that represent the corners of the voxel. Using the coordinate input, we also determine at which point along each dimension should we interpolate

to obtain the final data point and finally put everything together into the formula. The output of this trilinear interpolation method is one single data point within the voxel.

1.1.2 Maximum Intensity Projection

Maximum Intensity Projection is a volume rendering method that projects the voxel with maximum intensity into the 2D image plane. We implemented it by extending the basic *slicer()* method provided in the skeleton code and utilizing trilinear interpolation function.

In the basic *slicer()* method, the image displayed in the viewing plane is projected from area in the center of the 3D object. With this function, we see the object as if it is sliced in the middle. In *MIP()* method, we need more than one sample slice in order to obtain the maximum intensity value across the object plane. It is done by taking some sample slices in the direction of the viewing ray. To compute sampling ray, first we create a vector that perpendicular from the view plane and also intersecting the center of the object. Then we will adjust the length of the vector to 2 times from its original length (thus it will cover the whole object) and use it as our viewing ray. In our default setting, we take 150 slices spread evenly along our viewing ray.

Every time we slice the object along the viewing ray, we take the maximum intensity of the voxel by applying this formula:

$$I(p) = \max_t(S_t)$$

For each pixel in the viewing plane, only the voxels with maximum intensity value is projected.

1.1.3 Compositing Ray Function

Compositing is the volume rendering method that combines an image with background to create appearance of fully or partial transparency[2].

Similar to MIP function, we implemented compositing function by modifying *slicer()* method, resample and trilinearly interpolate the voxel along the viewing ray. But instead of taking the maximum intensity, the voxel color and opacity are computed. Then, the resampled colors and opacities of the foreground and background are merged with each other [3].

The formula below is applied to obtain the final value projected in the pixel, where c, α, and n denotes color, opacity, and number of sample, respectively:

$$I(p) = \sum_{i=0}^{n-1} c_i \prod_{j=i+1}^{n-1} (1 - \alpha_j)$$

The red, green, and blue component of the color is calculated separately. Therefore, each color component can be adjusted to highlight certain features of the object. The opacity is recalculated for each pixel x_i using the following equation.

$$\alpha_{tot}(x_i) = 1 - \prod_{n=1}^{N}(1 - \alpha_n(x_i))$$

1.1.4 Responsiveness Improvement

The raycaster software become significantly slow as the computational complexity increased. We tried to improve the responsiveness of the software by adding a limit to the number of sample processed for each intensity-based rendering type.

1.2 Data Exploration

We applied MIP function to four different datasets to see how MIP function works in objects with distinguishable characteristics. Figure 2 displays how MIP function works in the selected datasets and figure 3 displays some datasets under Compositing function.

1.2.1 Maximum Intensity Projection

Since MIP takes the maximum intensity of each voxel in every slice along the viewing ray direction, we can see in the image plane the object which contains the highest intensity value regardless of its actual position within the object plane. For example, in the human body dataset, the bone is clearly visible even though it is located inside a body due to its higher intensity value. The same thing goes for the backpack where the small, solid objects is very clearly distinguishable. However, when the object has almost equal intensity value like in the case of orange and piggy bank, it could be hard to distinguish the front and back side of the object.

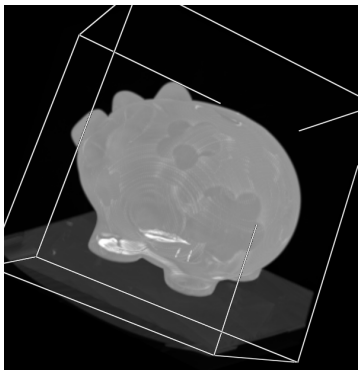
In terms of rendering time, MIP is significantly slower than the basic slicer method because more computation is done to visualize the dataset. The size of the object also affect the rendering time. Typically the bigger and denser the object, the higher the rendering time would be.

1.2.2 Compositing Ray Function

Unlike MIP which only project the voxel with maximum intensity value, compositing function takes into account all of the voxel intensity value, and assigns color and opacity into them.

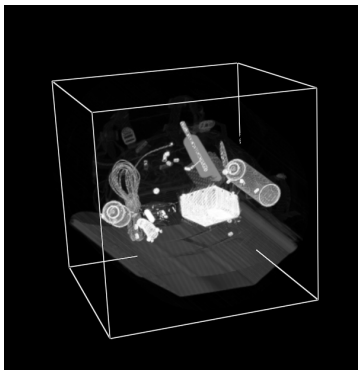
Since user can modify the color and opacity for particular intensity range through the transfer function interface, we can somehow distinguish an object from its surface. It is possible to emphasize certain feature of an object, like the bone of Carp and the Orange skin. However, we could not determine which feature in the object does an intensity is belong to, so when the supposedly different feature (in different layer) has the same intensity value, it would appear as the same in the viewing plane.

Compositing function requires more complex computational resource and time, marked by significantly higher rendering time ranging from 1099 ms to more than 8000 ms for bigger

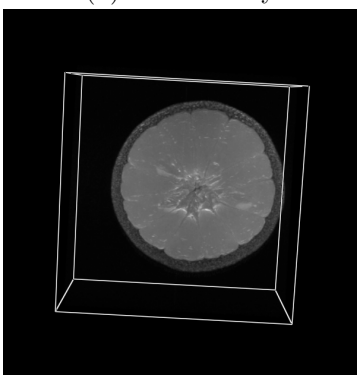


(a) pig

(b) human body

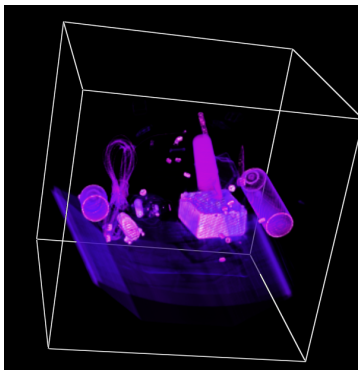


(c) backpack

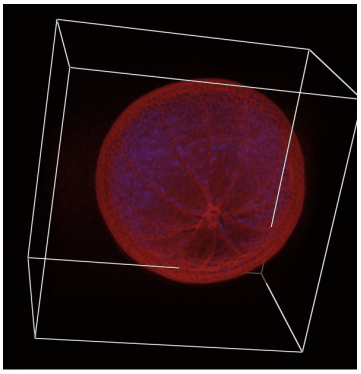


(d) Orange

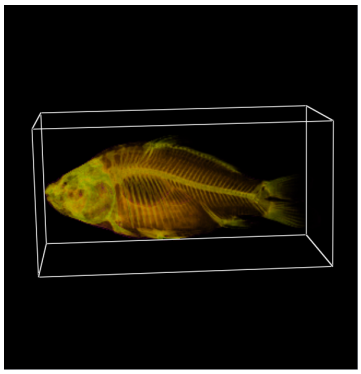
Figure 2: Result Illustration of MIP function



(a) backpack



(b) Orange



(c) Carp

Figure 3: Result Illustration of Compositing function

dataset. Human body dataset takes very long time to render, so another lighter dataset is used to see the effect on similar object type.

1.2.3 Responsiveness

We tested the MIP and Compositing function with different number of slice sample, ranging from 100 to 400, evenly distributed samples. We find out that as the number of sample

grows, so does the rendering time. This is not much of a problem in MIP function, but for another function that requires more complex computational task, it takes very long time to load.

From this experimentation, we conclude that 150 sample is a good range, with reasonable rendering time and good image quality.

Table 1: Rendering Time of MIP on different number of sample

Dataset	Sample	Time (ms)
pig8.fld	100	934
pig8.fld	150	4247
pig8.fld	200	4390
pig8.fld	400	5524

Table 2: Rendering Time of Compositing on different number of sample

Dataset	Sample	Time (ms)
pig8.fld	100	5324
pig8.fld	150	6048
pig8.fld	200	6344
pig8.fld	400	> 6344

1.3 Result

Among the volume rendering method that were tested, Slicer has the lowest rendering time because it has the fewest sample plain to compute. It is good to visualize certain section of an object, but it does not display the full feature of the dataset.

MIP is very good in visualizing the structure of an object with high density or hardness properties, which represented by color intensity. It is beneficial when the intensity difference of the object is quite high. However, it does not give a good sense of depth when the intensity of the voxel is almost uniform. Another disadvantage of MIP is that we do not have much freedom in deciding which feature of an object to be emphasized. We can only assign one color to the maximum voxel intensity.

Compositing function gives better, more realistic visualization on 3D object by allowing user to determine the color and opacity of certain intensity range, although at the cost of higher rendering time. More sampling would give better result, but the computational complexity also increases. Compositing function has higher rendering time among the other basic intensity-based rendering technique.

2 2D Transfer Function

2D transfer function is an improvement of the simple intensity-based transfer function. It allows the visualization of specific characteristic of an object, such as surface shading,

specularity, and material boundary.

2.1 Implementation

2.1.1 Gradient-Based Opacity Weighting

The typical isovalue contour surfaces renders all voxels with the value that exceeds certain threshold of f_v is rendered opaquely, while assigning zero opacity (full-transparancy) to the other area. But it still poses a problem that prevent multiple semitransparent surface from being displayed, even after applying window threshold, and produces some artifacts that are not present in the data. Gradient-based opacity weighting tries to improve the isovalue contour surface rendering technique by assigning opacity close to α_v for every voxel that has value close to f_v.

In order to calculate the suitable opacity value, we applied this formula:

$$\alpha(x_i) = \alpha_v \begin{cases} 1, & \text{if } |\nabla f(x_i) = 0| \; and \; f(x_i) = f_v \\ 1 - \left| \frac{1}{v} \frac{f_v - f(x_i)}{|\nabla f(x_i)|} \right|, & \text{if } |\nabla f(x_i) > 0| \; and \; f(x_i) - r|\nabla f(x_i)| \leq f_v \leq f(x_i) + r|\nabla f(x_i)| \\ 0, & \text{otherwise} \end{cases}$$

Where $\nabla f(x_i)$ denotes gradient vector, as defined below:

$$\begin{aligned} \nabla f(x_i) = \nabla f(x_i, y_j, z_k) \approx \; & [\frac{1}{2}[f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)], \\ & \frac{1}{2}[f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k)], \\ & \frac{1}{2}[f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1})]] \end{aligned}$$

To compute total opacity of a pixel obtained from each of the sampling voxel, we do the compositing. For the color, we use the color in the 2D transfer function editor color as our surface color.

Technically, we implemented this function extending the *GradientVoume* class. We let the user decide the minimum and maximum value for f_v (as 'Intensity' input in the 2D transfer function) and α_v (opacity), to give the user some freedom to determine which surface they want to emphasize.

Separation between different type of surface can be guaranteed if the dataset meet theses assumptions, as stated on Levoy's paper [3]:

1. The number of object types having CT number that falls within small neighborhood of some known value
2. Tissue of each object touch the tissue of at most 2 other types
3. If the objects are ordered based on their CT numbers, each objects only touches the object adjacent to it

2.1.2 Extended Triangle Widget

A basic triangle widget function basically selecting a region from intensity range around f_v that will be projected in the view plane. Extended triangle widget enables an option to set the maximum and minimum value of gradient magnitude $\nabla f(x_i)$. On the VolVis application interface, it extends the basic triangle widget implemented in gradient opacity weighting scheme by adding a column where user can set the lower and upper limit of the gradient magnitude[4] (Figure 4). The final image is rendered from the voxels within the triangle area between the point in data value (intensity) axis and two points that represent the radius and inside the range of desired gradient magnitude.

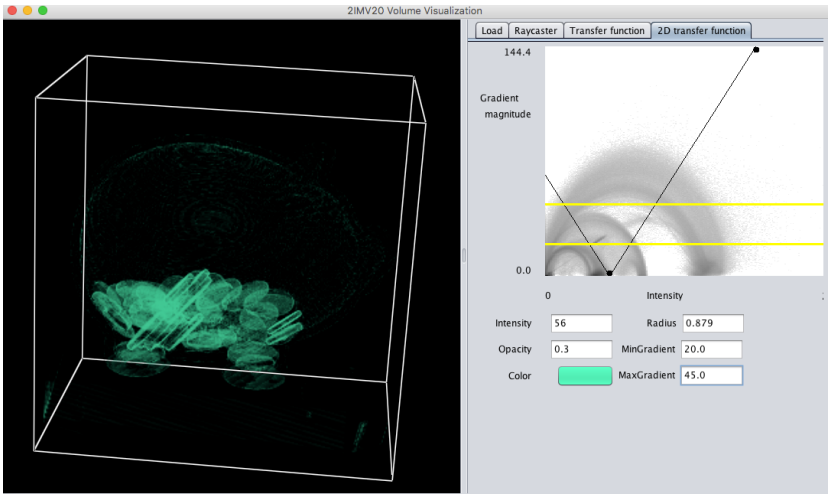


Figure 4: Extended Triangle Widget Interface

2.1.3 Illumination Model

The illumination model that we implemented for this assignment is Phong Shading Model. It is a method to give a smooth shading, sense of depth, and highlight by adding ambience, diffusion, and specularity to the 3D object. In this application, we implemented basic Phong Model, as formulated below:

$$I(p) = I_a + I_d k_{diff}(L \cdot N) + k_{spec}(N \cdot H)^{\alpha}$$

Where H is :

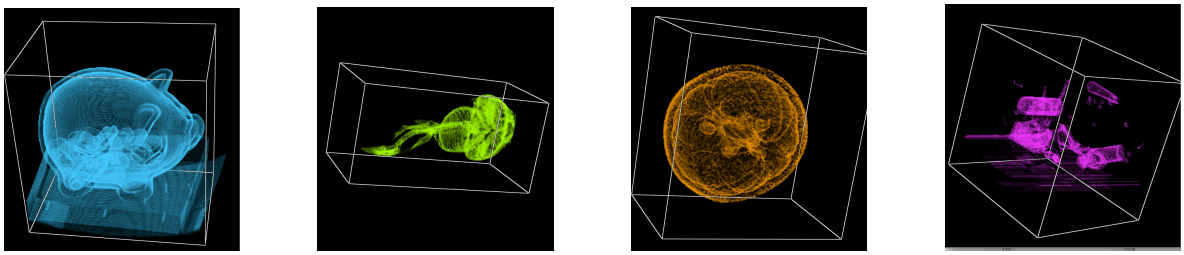
$$H = \frac{L + V}{|L + V|}$$

Here, we assume that the light source is white and comes from the the direction of user's viewpoint (thus, $L = V$), and we use the surface color obtained from transfer function for I_d. Also, we use constant $k_{ambient} = 0.1, k_{diff} = 0.7, k_{spec} = 0.2, = 10$.

2.2 Data Exploration

2.2.1 Gradient-Based Opacity Weighting

In testing this function, we used 2D transfer function menu to adjust the intensity, radius, and opacity of the object to be displayed. The way Gradient-opacity weighting work is different from the basic intensity-based rendering, in the sense that we decide first the intensity and range of voxels (that falls around that intensity) to be displayed instead of going through each voxel samples. Figure 5 illustrates the image result obtained from applying gradient-based opacity weighting function across several datasets.



(a) pig (b) Carp (c) Orange (d) Backpack

Figure 5: Result Illustration of 2D Transfer - Gradient Opacity Weighting Function

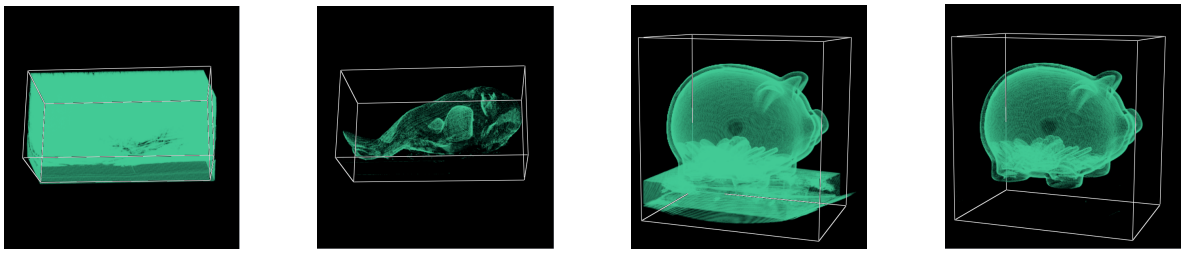
From our exploration, we find different quality of image produced by applying this function. The pig, carp, and orange show a good distinction between the different surface or tissue types, while the backpack is not as good. This might be due to the difference in quality of the dataset, in terms of CT number ordering. The potential causes of this problem is there are a lot of similar arc that overlap each other from the 2D transfer function of backpack dataset, which means the intensity and gradient magnitude for every item inside the backpack is similar, thus it is hard to separate each of them using extended triangle widget. We also find that it is possible to have a good distinction in certain intensity range while not as good in another range.

The radius value determine how many voxels around one intensity value to be rendered. The higher we set the radius, the richer the image would be. Setting the radius too high would allow more noise in the final image, so finding a good number is important.

2.2.2 Extended Triangle Widget

Adding extended triangle widget allows us to set the minimum and maximum value of gradient magnitude in addition to the radius variable in gradient-based opacity-weighting function. To better observe how this function work, we used some datasets that contain quite high noise in voxel-dense area.

From the 2D transfer function plot, we can predict that an object might consists of many part by looking at the arch of voxel density on certain intensity value. Modifying the minimum and maximum desired gradient magnitude and radius value allow us to highlight only certain part of the object.



(a) Carp at Gradient[0,173], r = 1,066, I = 37

(b) Carp at Gradient[25,100], r = 1,066, I = 37

(c) Pig at Gradient[0,144], r = 1,066, I = 9

(d) Pig at Gradient[25,144], r = 1,066, I = 9

Figure 6: Result Illustration of 2D Transfer - Gradient Opacity Weighting Function

It is shown that assigning minimum gradient magnitude over the highly voxel-densed area reduces the noise significantly. Figure 6 shows how setting different minimum and maximum gradient would affect the image. We find that setting higher minimum gradient magnitude gives more impact than lowering maximum gradient magnitude. In Carp dataset, setting gradient to [0,90] does not seem any different from [0,173]. But setting the magnitude to [25,100] significantly reduce the noise, as we can see in figure 6(b). Application on pig dataset also shows the same pattern. From those observations, we see that noise tends to occur at lower gradient magnitude.

Sometimes, if we set the gradient magnitude range too narrow, the image would lose quite a lot of important details. So considering trade-off between the radius and gradient magnitude is important to get the best result.

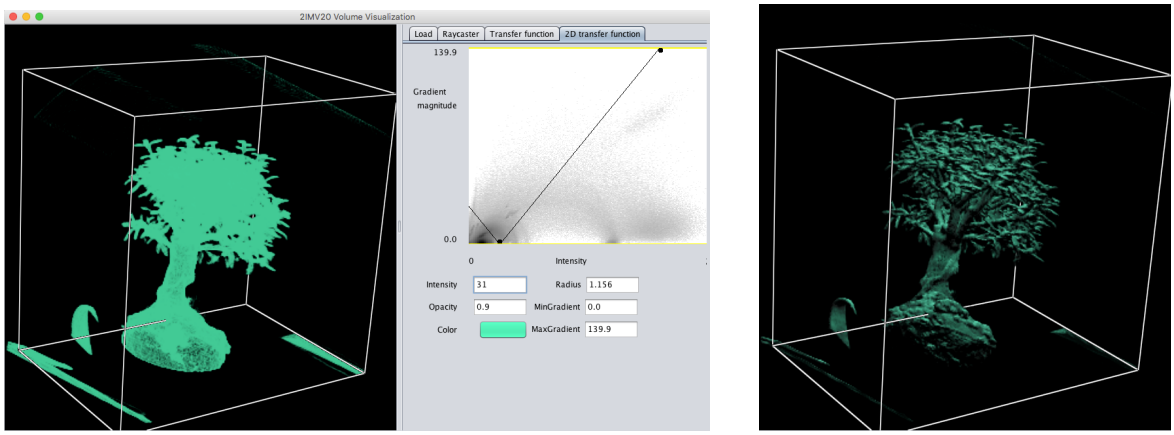
2.2.3 Local Illumination

To test this feature, we applied the Phong Shading function to bonsai.fld dataset and set the intensity value, opacity, and radius at 31, 0.9, and 1.156, respectively. The comparison between image without and with Phong Shading is shown on Figure 7. The image with Phong Shading appears to have more depth due to the shading and lighting applied for each voxels. It also gives more 'solid' touch to the image.

Gradient-based opacity weighting with Phong Shading requires very high processing time. This is because the function computes using Phong formula for each color and computes opacity of all of the voxels and combine all of them using compositing function. Meanwhile, on Gradient-based opacity weighting without Phong Shading only compositing opacity for each voxels. Figure 8 shows the application of shading on different rendering technique.

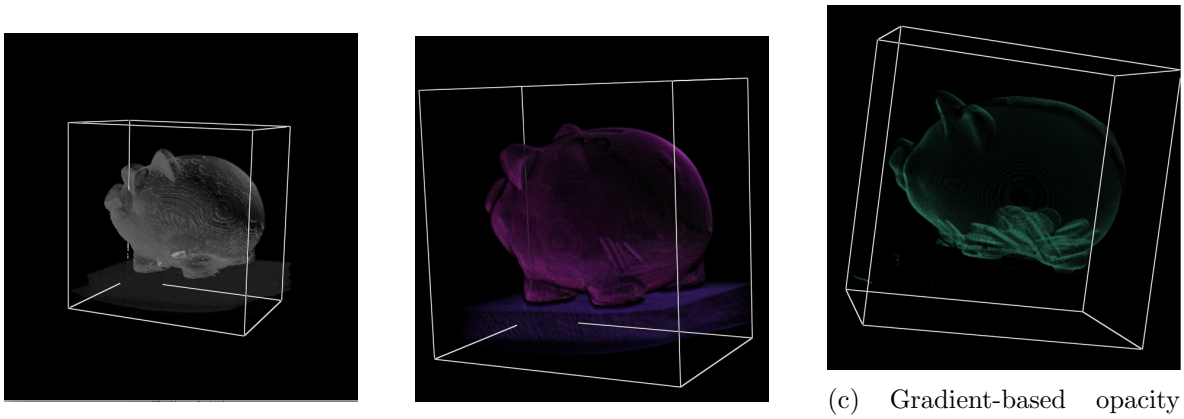
2.3 Result

Gradient-based opacity weighting works well for an object which has well-separated voxel value for different tissue type. It is good for quickly discovering the general structure of an object. User can also adjust the richness of the image by modifying the radius, although the overall image quality produced by gradient-based opacity function is lower than the one



(a) Bonsai without Phong Shading (b) Bonsai with Phong Shading

Figure 7: Result Illustration of Phong Shading



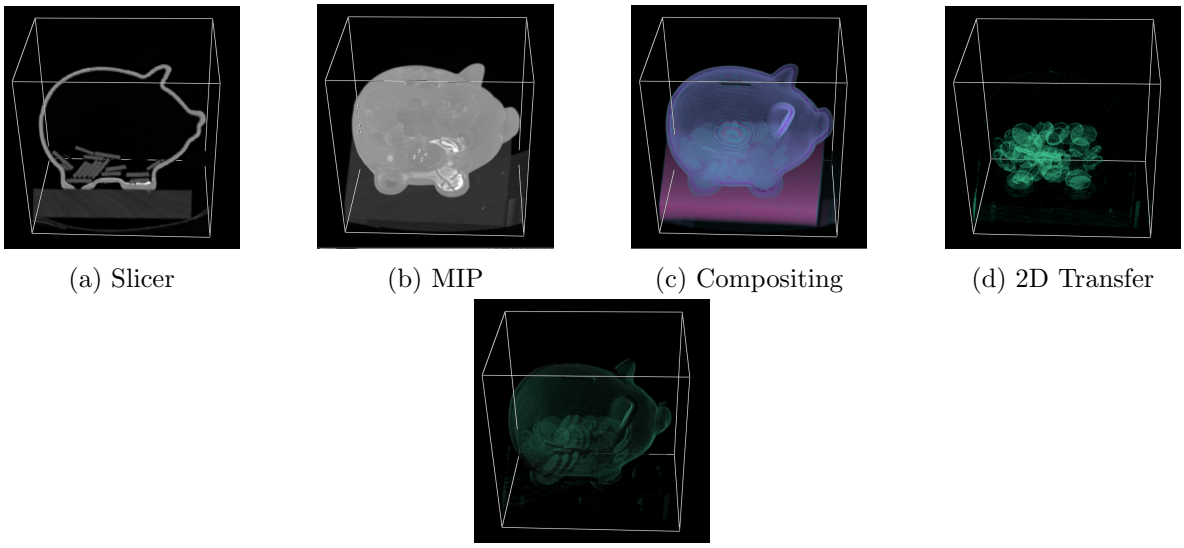
(a) MIP (b) Compositing (c) Gradient-based opacity weighting

Figure 8: Result Illustration of Phong Shading on other rendering function

from compositing function. One drawback of this function is that it is quite prone to noise, especially on wider radius. We must try to find the right intensity with good number of voxels at certain gradient magnitudes which allows less noise. If the data source is not well-structured, it could be hard to find the right combinations.

Extended triangle widget function compliment the gradient-based opacity function by allowing user to set maximum and minimum gradient magnitude at any point along the intensity range. This would be an advantage, especially if the quality of data source is unknown and we still want to highlight a feature from noisy area. It is also good to localize only certain part of an object, like the coin inside the piggy bank. However, the quality can only be as good as how the voxels are rendered at the gradient magnitude range we choose.

Figure 9 illustrates the comparison of different direct volume rendering technique on the pig8.fld dataset. Slicer method only shows the middle cross-section and does not give a full structure of the object. MIP highlights part of the object with highest intensity value regardless of their position in the scene and provides clearer overall structure than Slicer



(a) Slicer (b) MIP (c) Compositing (d) 2D Transfer

(e) 2D Transfer with Shading

Figure 9: Result Illustration of 2D Transfer - Gradient Opacity Weighting Function

function. Compared to MIP, Compositing function displays better layering effect and sense of depth so that the position of object's inner part can be made visible. 2D Transfer function, especially with extended triangle widget, is not as smooth as Compositing function, but it can separate components of the object from the whole unit. Local Illumination function adds shading and lighting effects to the object, giving an impression of solidness and depth.

Each direct volume rendering technique has its own advantage and disadvantage. Certain types of object might work well in one technique and not in another. But to determine which technique is the best, it depends on what kind of information we would like to extract from the dataset.

References

[1] Trilinear Interpolation
https://en.wikipedia.org/wiki/Trilinear_interpolation

[2] Alpha Compositing
https://en.wikipedia.org/wiki/Alpha_compositing

[3] M. Levoy. *Display of surfaces from volume data* IEEE Computer Graphics and Applications, vol. 8, no. 3, pp. 29-37, May 1988.

[4] J. Kniss, G. Kindlmann and C. Hansen. *Multidimensional transfer functions for interactive volume rendering* IEEE Transactions on Visualization and Computer Graphics, vol. 8, no. 3, pp. 270-285, Jul-Sep 2002.