Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science

Volume rendering assignment

2IMV20 - Visualization

Students:
Jose Luis Diaz Mendoza (1287338)
Juan Villaseca Gomez (1379127)

Eindhoven, December 2018

Contents

Chapter 1

Introduction

In general, Visualization techniques are important to discover patterns in the data in which statistical measures fail to expose, illustrated by Anscombe's quartet. In more concrete scenarios or domains such as medicine, visualizations and their interpretation are one of the key activities to reach successful results.

In this document, we will focus in 3D volume rendering techniques based on a raycasting approach, to transform 3D volume data into 2D projections. These techniques are slicing, maximum intensity projection (MIP), compositing using 1D transfer function and 2D transfer function. The purpose of the document is to explain how these techniques have been implemented. Furthermore, we will apply these techniques to the Deep Water Asteroid Impacts Dataset (DWAI dataset)[1] provided by the 2018 visualization challenge[2].

The structure of the report is as follows: section 1 explains at a high level what are the techniques, our implementation guideline and, the details in which we struggled to detect and solve. In section 2, we try to provide some meaningful insights about the DWAI dataset. Finally, we compare the different techniques, in terms of, strengths and weaknesses and we discuss other visualization parameters, i.e rendering time and responsiveness.

Chapter 2

Visualization techniques

The project base was given with basic components already built on: user interface and the slicing technique were already implemented. The slicing technique takes a slice of the volume and projects it in the view plane.

To fulfill the requirements, the following work pipeline has been followed, where the function applied is slicing, MIP or compositing using 1D and 2D transfer functions:

Figure 2.1: Visualization Work pipeline

2.1 Raycasting

To be able to transform 3D volume data into 2D images, techniques such as raycasting or raytracing are needed. Raycasting tries to emulate the human eye by casting light rays along the volume.In this way, the rays goes through the volume and the samples (voxel values) taken along the ray are transformed by using a function and projected into the view plane to form the final image.

As a convention, the image height will be defined by the Y axis, the width by the X axis, and the depth by the Z axis. Furthermore, the view plane (composed by orthogonal unitary vectors uVec ,along the X, and vVec, along the Y axis) is centered in the center of the volume. Another vector (viewVector, along the Z axis) orthogonal to both uVec and vVec has been defined and is the one which indicates the direction of the ray.

In order to implement the ray, the slicing technique has been modified to extend it. The major change has been to include a loop which go through the z dimension and take samples along the way. To do so, we first need to get the coordinates of each voxel along the ray and get the value associated with that voxel.

There are few aspects to be taken into account:

1. The maximum length of a ray is the one which goes through the diagonal of the volume. Thus, we need to iterate over the number of values of the diagonal of the volume, making sure that in all the cases we are taking into account the values of the whole volume and not leaving any voxel out of consideration. Using this approach, in the cases where we dont iterate over the diagonal of the volume, we compute the function in empty spaces where there is not volume, which doesnt affect to the computation.

2. Furthermore, the view plane is in the center of the volume. Since this is our reference, half of the coordinates are positive and the other half, negative.

3. To map the pixels of the view plane to the voxel coordinates, we added the z component to translate the coordinates.

These details are shown in the pseudocode below:

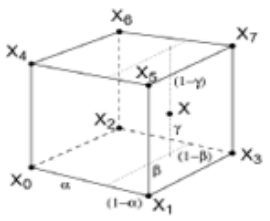
---

for $j = 0 \rightarrow image.Height$ **do**
  for $i = 0 \rightarrow image.Width$ **do**
    for k = - diagonal/2 $\rightarrow$ $diagonal/2$ **do**
      pixelCoord = **uVec** * (i - imageCenter) + **vVec** * (j - imageCenter) +
        + **viewVec** * k + **volumeCenter**

Where **pixelCoord**, **uVec**, **viewVec** and **volumeCenter** are vectors with 3 components and *image* is the 2D projection where we display the result pixel.

2.1.1 Trilinear interpolation

Once the pixel coordinates of the voxels are computed, the scalar values of those voxels are needed. The problem is that the data does not provide values for every coordinate in the volume, just the ones of the corners of each voxel (see figure below). For that reason, an interpolation in the 3 axes is needed to estimate the value of any coordinate inside a voxel (out of the ones which the data provides). We have follow the trilinear interpolation formula to solve this issue:



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

Where $\alpha = \dfrac{x - x0}{x1 - x0}$, $\beta = \dfrac{x - x0}{x2 - x0}$ and $\gamma = \dfrac{x - x0}{x4 - x0}$

Figure 2.2: Trilinear interpolation formula

As we have introduced above, once the samples and ,then, its values are taken along the ray, they need to be summarized. This implies that a function is needed to reduce all values into one, in a meaningful way. Two possibilities of those functions have been implemented: MIP and compositing.

In our code we have also considered the situation when you have a value of the corners of the voxel, where no iteration needs to be applied, these means returning a 0 for the corresponding parameter alpha, beta or gamma.

```
double alpha = ((coord[0] == xLow) ? 0 : (coord[0]-xLow)/(xHigh-xLow));
double beta = ((coord[1] == yLow) ? 0 : (coord[1]-yLow)/(yHigh-yLow));
double gamma = ((coord[2] == zLow) ? 0 : (coord[2]-zLow)/(zHigh-zLow));
```

In the previous formulas, the Low and High coordinates are computed by applying a round to lower value (*Math.floor*) or higher value (*Math.ceiling*) operators.

2.1.2 MIP

MIP consists in keeping the maximum value of the samples of each ray. Therefore, after retrieving the scalar value out of the 3 coordinates of a voxel for each sample, the only step left to do is computing the maximum value of all the samples of the ray.

While being inside the 3 loops mentioned in the Raycasting section, we compute the pixel value of the given coordinate and store for each ray the maximum value, which is the one we will project in the 2D image.

```
val = interpolateVoxel(pixelCoord);
if(val > max_val){
    max_val = val;
}
```

It is worth mentioning that MIP will yield symmetric 2D images from one view perspective and the opposite one, since its computing the maximum voxel value per ray. In other words, MIP will always highlight the maximum value of each ray (see figure below).

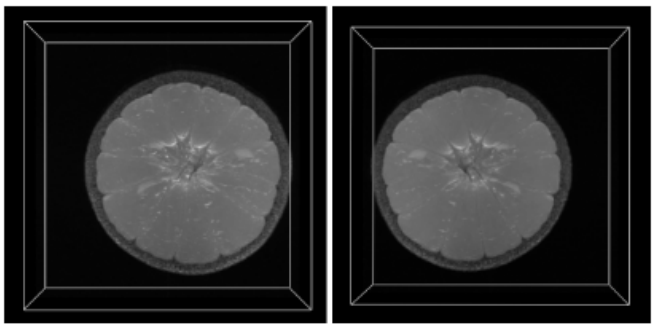


Figure 2.3: Comparison between two MIP images from opposite sides

2.1.3 Compositing

Compositing method maps each voxels scalar value to a color and opacity values by using a transfer function. The aim of compositing method is to show more clearly surfaces in the volume. The method works as follows:

1. We start retrieving the opacity value of each voxel from the backside of the volume to the front (moving in Z direction along the ray).

2. Each opacity value, corresponds to a color. This transformation is made by the transfer function.

3. While moving along the ray, we integrate all the opacity contributions, obtaining in the end a composite color and opacity of each ray.

The value composition is developed in the code by using two variables: *voxelColor*, which computes the corresponding color of the current voxel by using the transfer function *tFunc*; and *compositeColor* which stores the composite color values of the previous voxels. By this, in each iteration, we update the *compositeColor* variable, obtaining in the end, the composite value of each ray.

```
voxelColor = tFunc.getColor(val);
compositeColor.r = voxelColor.r * voxelColor.a + (1.0 - voxelColor.a) * compositeColor.r;
compositeColor.g = voxelColor.g * voxelColor.a + (1.0 - voxelColor.a) * compositeColor.g;
compositeColor.b = voxelColor.b * voxelColor.a + (1.0 - voxelColor.a) * compositeColor.b;
```

The transfer function allows to highlight more or less the differences in the opacity for each scalar value. In the following figures it is shown this behavior:

- Transfer function given by the original code.

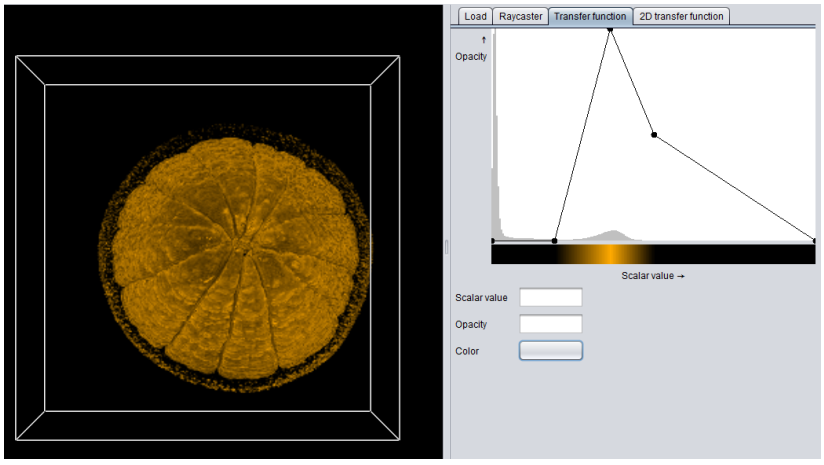


Figure 2.4: Compositing using original 1D transfer function

- Transfer function that follows the opacity distribution, filtering the highest opacity values of the left side (colored in black). As we can observe, since theres no big differences in the opacity values of the transfer function, the color of the projected image is quite homogeneous and doesnt highlight properly the surface.

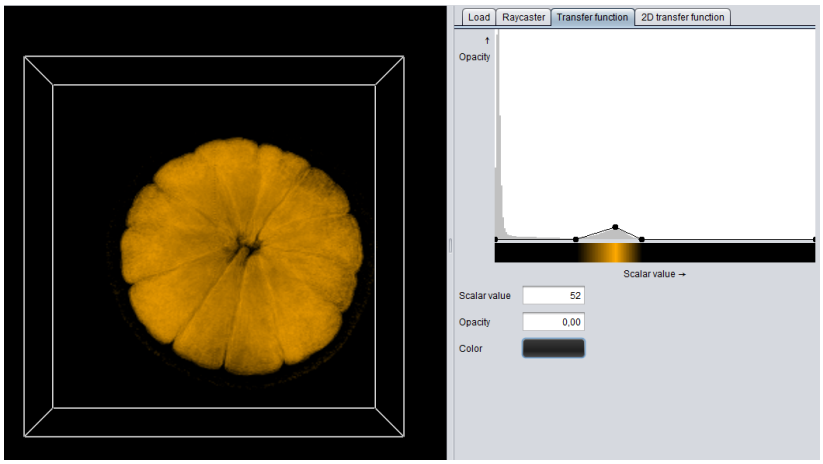


Figure 2.5: Compositing using a lower maximum opacity 1D transfer function

- In this case instead of just following the opacity distribution, as in the previous case, we give the maximum possible value opacity (opacity = 1) where there is a maximum in the distribution. By this, we amplify the original opacity value of the distribution, making bigger the difference in opacities and therefore highlighting better the surface. The result is similar to the first case with a bit more contrast.

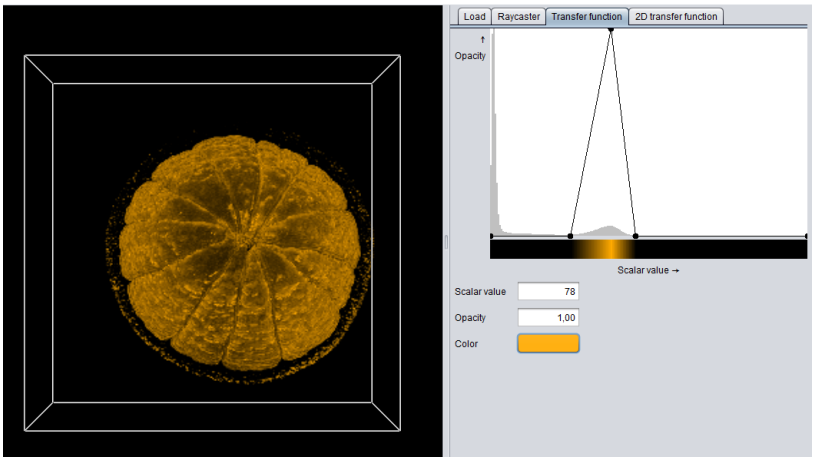


Figure 2.6: Compositing using a higher maximum opacity 1D transfer function

2.1.4 Responsiveness

Since the computation of the voxel values along the ray requires considerable amount of computing power, mainly due to the trilinear interpolation that needs to be applied to each coordinate, we have implemented a way to make the application more responsive, specially when moving the cube point of view. Without taking this into account, while moving the point of view with the mouse, the program would compute the rendered image for each of the new intermediate point of views. This makes the application working really slow.

To avoid this situation, when the mouse is pressed and the user is rotating the cube, the rendered image is computed taking larger steps along the ray (steps of 10) and therefore, lowering the computing time needed. It is just when we stop rotating the cube and we release the mouse, when the rendered image is computed taking all the values of the ray (steps of 1). In the code, the boolean variable *interactiveMode* is the one that controls if the mouse is pulsed or released. Depending on the value of this variable, we use different steps:

```
int stepInteractive = 10;
int step = (interactiveMode) ? stepInteractive : 1;
...
for(double k = -diagonal/2; k < diagonal/2; k+=step) {
...
```

This can be visualized in the following images. As it is shown, in the right image (*interactiveMode* = 1), when we are rotating the cube, the steps to render the image are larger and we dont consider all the voxel values of the ray. Therefore, the rendering time is much lower than the one of the left image (*interactiveMode* = 0), where the image is rendered considering all the values of the ray.

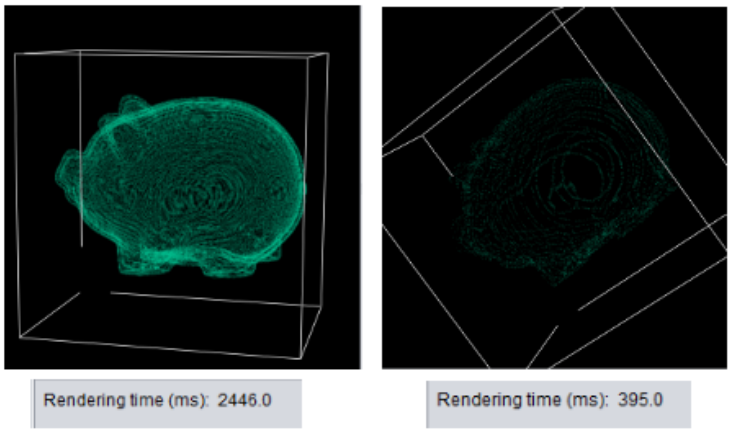


Figure 2.7: Comparison between 2D images rendered without and with rotation of the cube and their respective rendering time

2.2 2D Transfer function

2.2.1 Gradient-based opacity weighting

Once composite technique is understood, we are going to apply a 2D transfer function using Marc Levoys formula [3]. The 2D transfer function maps the color and opacity according to the scalar value of the voxels but also according to their gradient. First, we implement a method to compute the gradient of a voxel. The gradient is computed by subtracting the value of the voxel with its closest neighbor. This measure how different it is from the other. In our implementation, the gradient is computed for each of the voxel in the volume as follows:

```
gradX = (gradientVolume.getVoxel(i+1,j,k) − gradientVolume.getVoxel(i−1,j,k))/2;
gradY = (gradientVolume.getVoxel(i,j+1,k) − gradientVolume.getVoxel(i,j−1,k))/2;
gradZ = (gradientVolume.getVoxel(i,j,k+1) − gradientVolume.getVoxel(i,j,k−1))/2;
```

However, for the same reason than the values of the voxels, an interpolation is needed but, applied to the gradient. Secondly, we implement Marc Levoys formula. In this way, the formula yields the opacity of the voxel when the value does not change (the gradient is equal to 0) and the value of the voxel matches a reference value, a intermediate value when the variation or gradient is within two boundaries also marked by the reference value and radius, and, 0 otherwise.The logic is illustrated in the pseudocode:

```
if ( magOfGrad == 0 and fv == valueOfVoxel):
    Opacity = opacityOfVoxel;
if  else (fv <> (magOfGrad ∗ radius) and magOfGradient > 0 )
    Opacity = opacityOfVoxel ∗ factor;
else :
    Opacity = 0
```

Where *fv* (the threshold) and *radius* (determine the amplitude of the boundaries) are input parameters of the program , and factor is defined as:

$$factor = \frac{1}{radius} * \frac{|f_v - valueOfVoxel|}{|mag|}$$

Finally, the compositing is applied as the compositing section explained.

2.2.2 Phong illumination model

The phong model is a reflection model based on three parameters, i.e the three illumination terms: diffuse (non-shiny illuminations and shadows), specular (bright, shiny reflections), ambient (background illumination). The model is empirical and it is expressed as:

$$I = I_a k_{ambient} + I_l k_{diff}(\mathbf{L.N}) + I_l k_{spec}(\mathbf{V.R})^{\alpha}$$

The ambient component does not change according to the angle of light rays with the surface, in comparison to the specular and diffuse components. In the formula, this is reflected by the (L.N) and (V.R) terms. The other variables are properties of the materials, i.e reflectivity of the surface.

In order to apply the phong illumination model, instead of using the standard formula, we have followed the guidelines of the simplified version:

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

In which,vector R has been replaced by H and vector V is estimated by computing the gradient and normalizing it (i,e dividing by its module). Vector L is the viewVec vector (which is defined by the light rays entering into the volume) but in the opposite direction as the L vector aims out of the volume. The implementation works under the following assumptions:

1. Vector L is equal to vector R, thus, equal to vector H.
2. I_d is the color of the surface yielded by the transfer function.
3. The formula does not apply when the dot products are positive.

Chapter 3

Contest dataset exploration

3.1 Explanation of the dataset

The Deep Water Asteroid Impacts Dataset (DWAI dataset) represents the study of asteroid impacts in deep ocean water. As the full dataset is too large to handle, only a subset of the data, containing a few time points and only the scalar fields v02 (volume fraction water), v03 (volume fraction of asteroid), tev (temperature in electronvolt), and prs (pressure in microbars) is going to be explored. Only these two types of datasets have been tested:

- **yA31**: A $\rightarrow$ No airburst, full asteroid impacts water, 3 $\rightarrow$ Asteroid of 250 meter of diameter, 1 $\rightarrow$ Asteroid entering with angle of 45.
- **yB31**: B $\rightarrow$ Airburst at elevation of 5 Km above sea level, 3 $\rightarrow$ Asteroid of 250 meter of diameter, 1 $\rightarrow$ Asteroid entering with angle of 45.

The meaning of the four-character name of these datasets is provided in the original contest paper[4].

Deep Water Impact Data Set Naming Convention
1st Character - No Intrinsic Meaning
2nd Character - Airburst [A] No airburst, full asteroid impacts water. [B] Airburst at elevation of 5 kilometer above sea level. [C] Airburst at elevation of 10 kilometers above sea level. [D] Airburst at elevation of 15 kilometers above sea level.
3rd Character - Asteroid Diameter [1] 100 meter diameter. [3] 250 meter diameter. [5] 500 meter diameter.
4th Character - Angle Of Entry [0] Asteroid initialized with 27.4 degree momentum. [1] Asteroid initialized with 45 degree momentum. [2] Asteroid initialized with 60 degree momentum.

Figure 3.1: Summary of the four character ensemble member naming convention.

Possible questions of interest that arises are:

a) Which of both is more dangerous?, i.e which produced a higher displacement of water.

b) Possible side effects such as abrupt variations of temperature or pressure could cause storms?

c) How the asteroid evolves over time?

However, from our exploration, we have decided to analyze the cases presented in next section.

3.2 Analysis of some interesting cases

3.2.1 Case 1: Events in yA31 have indeed small time interval between them

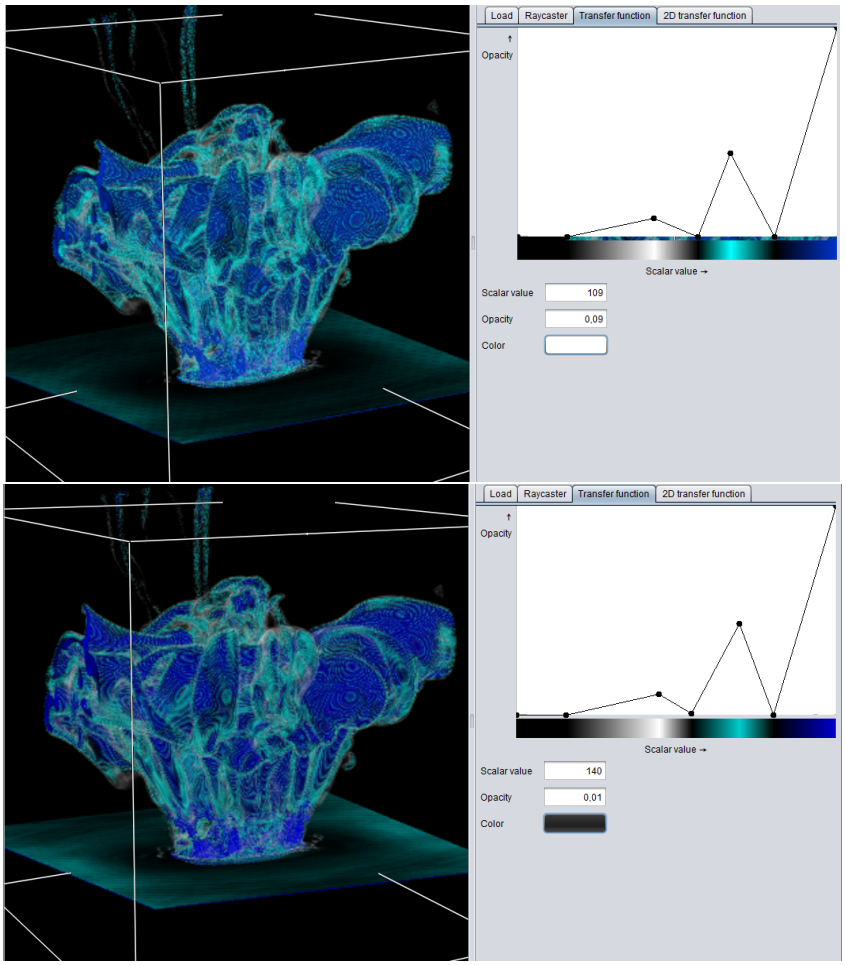


Figure 3.2: Comparison of the volume fraction of water between first (39476, top) and last event (39803, bottom) of yA31.

In this case, we wanted to asses the change of the water volume distribution and analyze the time interval between events of the yA31 dataset. We have applied to the initial (39476) and final (39803) events the same transfer function and we rendered the 2D image from the same point of view. As it is shown in the figure below, there is an indistinguishable difference between the two water distributions, so we confirm that the time window between the events in this dataset is quite short.

3.2.2 Case 2: Evolution over time of asteroid yB31 making impact in water

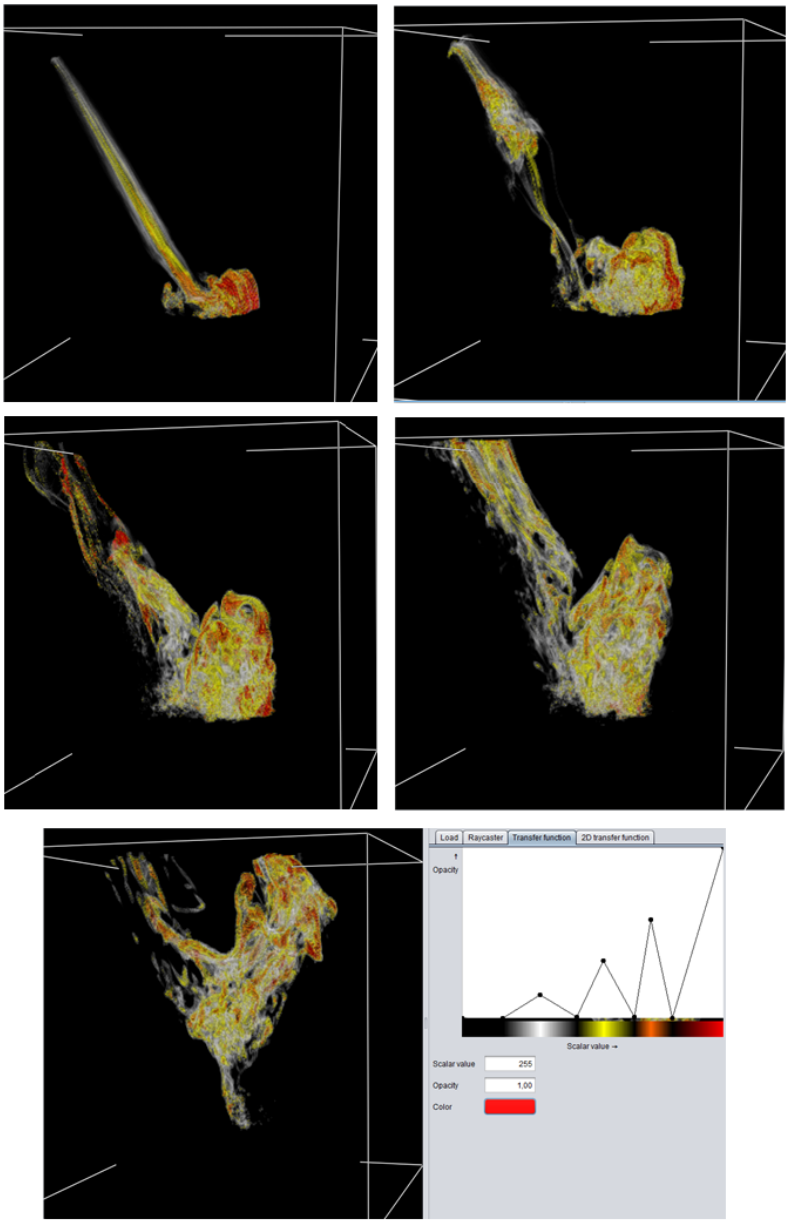


Figure 3.3: Volume fraction of asteroid of all events of yB31.

In this case, our aim was to observe the asteroid of yB31 over the different time frames. We wanted to follow the falling and impact process. In the sequence of images, we can appreciate the different concentration distribution of asteroid's volume. In the first moment, the asteroid is more compact and its volume is concentrated in the front part (shown in red). As the time passes, there is a scatter trend, in which the red tone is less noticeable as it is increasingly scatter around the volume. To reach this visualization, we have selected four ranges of values and assigned them an increasing opacity. The four regions are separated to black strips to be clearly delimited. Moreover, to make the visualization of the sequence more easy to follow, we have just plotted once the transfer function that is used in all cases.

3.2.3 Case 3: Relationship between asteroid fraction volume and its temperature

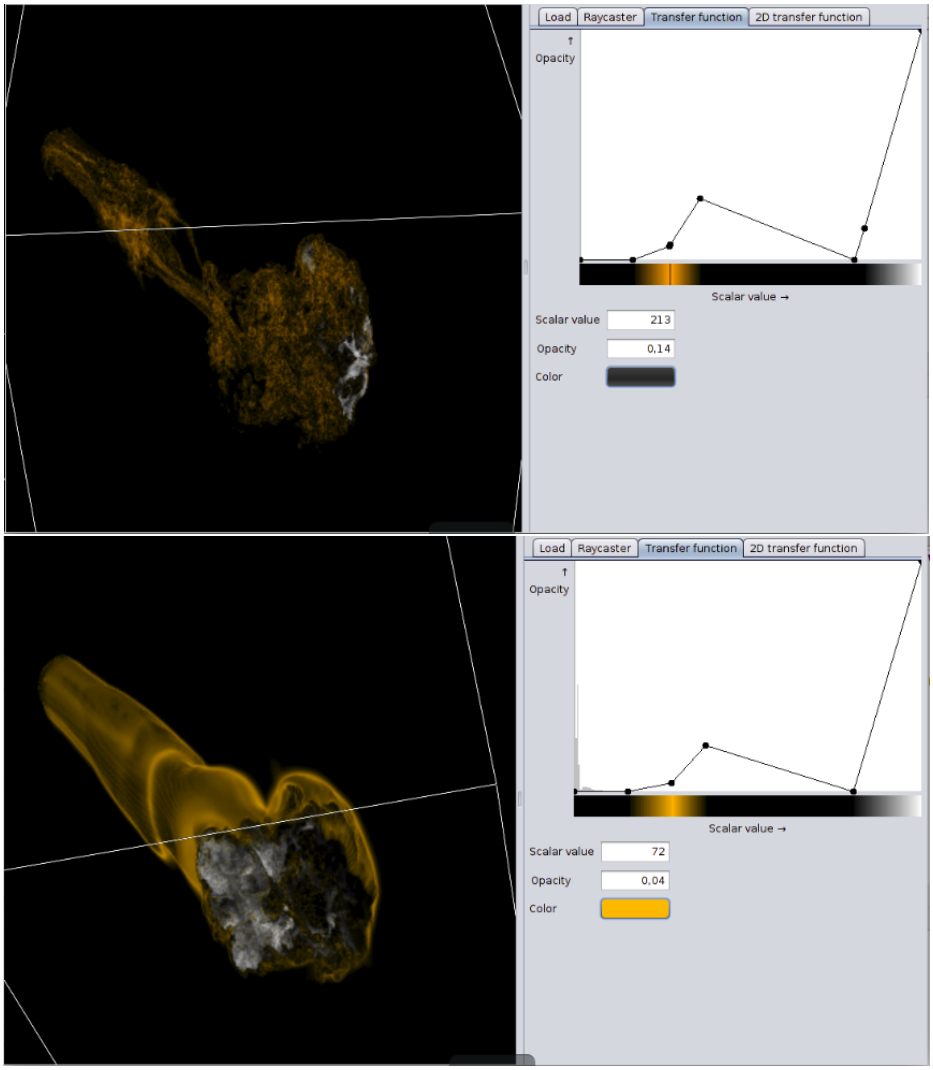


Figure 3.4: Comparison between the distributions of the asteroid volume fraction (top) and its temperature (bottom) of event yB31-21302.

In this case, our main goal is to find a correlation between the temperature and the volume of the asteroid. Our hypothesis was that the asteroid volume is positively correlated to the temperature. This assumption is made since we supposed that the inner part of the asteroid would be warmer and the volume concentration would be higher. In order to rule out the hypothesis, we thought that that the most warm and concentrated areas should be highlighted, thus, we assigned the maximum opacity (color white) and to show some contrast, we added the second orange component. Firstly, we encountered that the warmest and most concentrated volume does not match. Secondly, it seems that the relatively low temperature wraps the asteroid and the warmest part is in the asteroid.

3.2.4 Case 4: Comparison between water fraction volumes of last events of yA31 and yB31 datasets

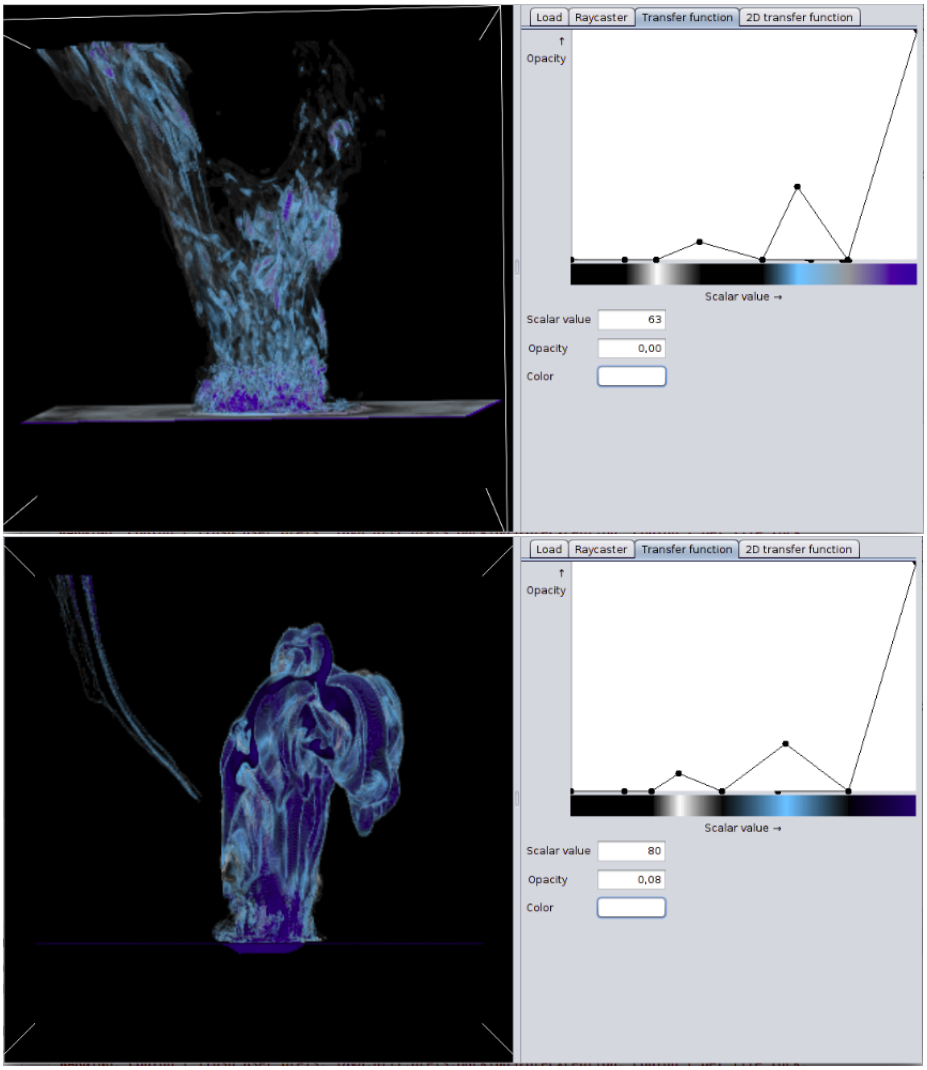


Figure 3.5: Comparison between the water fraction volumes of yA31-39803 (top) and yB31-44444 (bottom) events.

In this case, we took the latest frame of both datasets to assess how the different types of asteroid (no airburst -yA31- and airburst -yB31- types) would impact into the ocean and, therefore, would pose a bigger threat in terms of water volume flooding a population. To do so, we have selected darker blue colors and high opacity to high water volume values while the opposite to lower values. As it is shown, the airburst figure (bottom) has more areas covered by high water volume values, while the asteroid with no airburst (top) just present these higher water volume values at the foot of the water mass and in some small and spread areas. Thus we deduce that the airburst asteroid poses a major threat than the asteroid without airburst.

Chapter 4

Evaluation

The different cases have shown how visualization could enhance human perception to discover hidden patterns or answer questions. The results have support or refute different hypothesis, which we would liked to test out. The transfer function discovery has proven to be tricky but our main strategy was to separate the regions which we want to highlight, as it were a classification problem. In the case of the water displacement, it was a cornerstone to assign lower opacity to lower scalar values to highlight the zones with more concentration of water. In the second case, we wanted to explore the asteroid volume evolution over time to monitor the different distributions of volume of the asteroid until the impact. In this way, the different amount of fraction volume had to be delimited to be able to appreciate the variation through time.

The different techniques exhibited have different purposes, depending on problem to solve. MIP is good technique when high intensity is needed to be highlighted compared to its surrounding voxels. For example: it is useful to highlight blood vessels, as it contrasts better between the high intensity blood against the surrounding tissue than summing up the entire background along the volume[5]. Another, strength is the straightforward implementation because none design process is required. Thus, it yields faster rendering times compared to compositing.

On the other hand, compositing with 1D transfer function allows to classify materials with different optical properties. Although, 1D transfer functions are not suitable for noisy datasets or overlapping materials datasets because when one data value belongs to different boundaries, a 1D transfer function is unable to isolate them properly[6]. Another remarkable drawback is introduced by the the transfer function itself. A trial and error approach is required in the designing process of it, difficulting the implementation. However, through a interactive widget it grants flexibility by tuning the color and opacity of the visualization.

Compositing with 2D transfer functions share the same purpose than the 1D transfer function compositing. In the sense, they also capture the distinction between different materials but they are also capable to detect complex combination of boundaries between multiple materials. In addition, they can deal not only with one value but the combination of several of them because dimensions can be added as needed. Thus, They can even show different aspects of the boundaries, such their thickness. The extra-capabilities allow more deep visualization in exchange of complexity. This affects the rendering time (increasing it) because further calculations are needed compared to both 1D compositing and MIP.

To sum up, we have implemented different 3D direct volume rendering techniques. We have explain the foundations of these methods and the implementation details. Then, these methods have been applied to the singular asteroid deep water impact dataset to provide some insights and, finally, the yielded results were commented and a, overall, comparison of the techniques was discussed.

Bibliography

[1] Scivis Contest webpage: https://sciviscontest2018.org/

[2] Deep Water Impact Contest webpage: https://oceans11.lanl.gov/deepwaterimpact/

[3] Levoy, Mark. "Volume Rendering: Display of Surfaces from Volume Data", IEEE Computer Graphics and Applications 8, 5. (May 1988)

[4] John M. Patchett and Galen R. Gisler, Deep Water Impact Ensemble Data Set. (April 2017)

[5] Charles M. Anderson, David Saloner, Jay S. Tsuruda, Lorraine G. Shapeero, and Ralph E. Lee. Artifacts in maximum-intensity-projection display of MR Angiograms. (1990)

[6] Joe Kniss, Gordon Kindlmann, Charles Hansen. Multi-Dimensional Transfer Functions for Interactive Volume Rendering. (2002)