CS677: TOPICS IN LARGE DATA ANALYSIS AND VISUALIZATION

Assignment 1 - Parallel Distributed Axis-aligned Volume Rendering using MPI

Report

By

Group-1

Parjanya Aditya Shukla - 241110046
Divyansh Chaurasia - 241110022
Ansh Makwe - 241110010

Email: { anshakwe24, cdivyansh24 } @cse.iitk.ac.in

Indian Institute of Technology, Kanpur

Code Flow

The program flow start with the processor having rank 0 receiving all inputs such as filename, partition type, step size, x_min, x_max, y_min and y_max. The x_min, x_max, y_min and y_max determines the coordinates of the region the user is interested for visualization. The file contains raw 3d data whose dimensions are declared in the filename.

Reading dimensions of 3D Data

The function **read_file_dimensions** that takes filename as the input, interprets the dimensions of the 3d data and prints the extracted dimensions. These dimensions are stored in the variable named as **dims.**

Reading the raw file

The function **read_raw_file** takes filename and dimensions dims as input parameters and read the data as floating point numbers from the filename using fromfile function of the Numpy library. It also formats data into the correct 3d structure as specified by the dimensions dims using the reshape function of the Numpy library. The formatted data is stored in the variable named as **data.**

Trimming the data

The function **trim_data** accepts data, x_min, x_max, y_min and y_max as arguments and filters out the data of the region which the user is interested in visualizing. The filtered data is stored in the varaible named as **trimmed_data.**

Partitioning the data to all processors

The function **partition_data** accepts trimmed_data, size {the number of processors }, partition_type as parameters and partition trimmed data approximately equally to all the processors thereby implementing parallelism. Each processor is responsible for performing the visualization task for the data assigned to it. The partition is performed either on one axis {x-axis or y-axis} or both axes { x-axis and y-axis } based on the partition type specified by the user. To partition the data we calculate the chunk size each processor should get. Also when decomposing data on both axes we ensured that the number of processors along x-axis are greater than the number of processors along y-axis. For example if the no. of processors N = 15, then processors along x-axis should be 5 and processors along y-axis should be 3 as 15 = 5*3. Factorization of N was done to split the no. of processors in the expected way. The decomposed data was stored in the variable named as **sub_data_chunks.**

Reading the opacity and color transfer function

The function **read_file_to_pairs** takes the filename of the opacity transfer function file and reads it to form pairs of data points and the opacity value at each data point. A list of such pairs is stored in the variable named as **transfer_functions.** Further the color transfer function is read globally and stored in **triplets_list.**

Transfer Functions

Opacity and Color Transfer functions are used in ray casting to calculate the color and opacity value at sample points along a ray. These transfer functions takes raw data values and maps them to color and opacity values. For the color transfer function there are actually 3 different transfer functions for red, green and blue values. These transfer functions are used to calculate the value at any sample point using linear interpolation for the data value.

Broadcasting info to all processors

We then use the concept of inter-process communication to broadcast the opacity transfer function from rank 0 processes to all processes. We also scatter the **sub_data_chunks** to all processes and broadcast the step size so that each process can perform ray casting.

Each process processing data

Each process receives the data for performing ray casting, resulting in a local image created at each process. These N local images of N process are later stitched together to create the final image. The **process_data** function takes sub_data_chunk, step_size, opacity_tf as arguments and internally calls the **ray_casting** function, with the same parameters. We perform ray casting along the z-axis and follows front-to-back compositing. We also implement early stopping in which we stop at the sample point where the opacity reaches the maximum threshold of 1. For each point along the z-axis, we calculate the value of data using linear interpolation and using this value we extract the color and opacity values using color transfer function and opacity transfer function. The color value is calculated using RGB values - 3 values each defining proportion of red, green and blue color in the actual color. The **apply_transfer_functions** and **opacity_ttf** performs the operations for color and opacity respectively.

Ray Casting

Ray casting is a technique to perform volume rendering for visualization of large data. In this technique the aim is to create a 2D Image of the data. This techniques involves dropping rays from the eye view to the data such that each ray corresponds to exactly one pixel of the final image. Along the rays we select few sample points and the value of the data at these sample points are calculated using the technique called **linear interpolation**. We also calculate the color and opacity value at these sample points using **back-to-front** and **front-to-back** rendering. We generally know the value of opacity and color at few points of the data and we use them to calculate the opacity and color transfer functions.

Linear Interpolation

Linear Interpolation is a technique to calculate the data value at any sample point using the data values of its neighbouring points. The general form of linear interpolation is

$$V_p = \sum V_j * W_j$$

such that p is the sample point, j is any point that belongs to the neighbourhood of p. The exact formula for linear interpolation depends on multiple parameters like cell type, data values at the cell corners and the positional coordinates of point p in the cell. In this assignment we perform linear interpolation on a line using the formulas -:

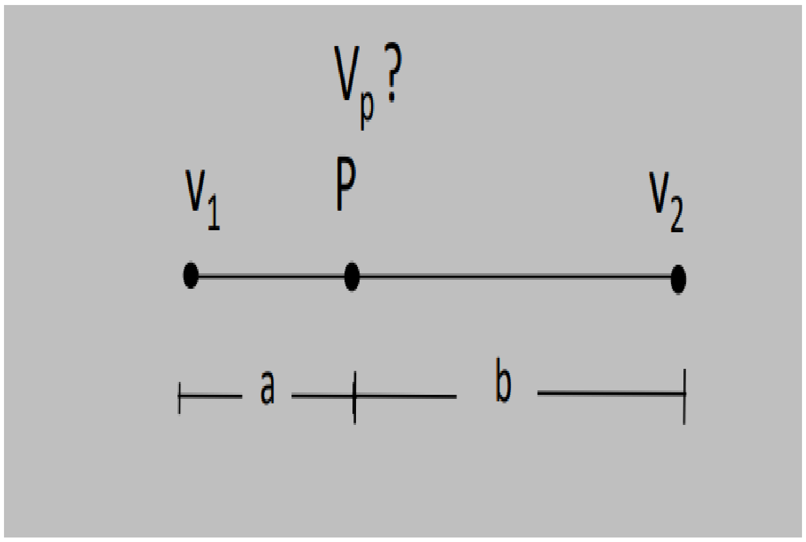

Figure 1

Position of point P

$$\alpha = a/(a+b)$$

Linearly interpolated value at point P

$$V_p = (1-\alpha) * V_1 + \alpha * V_2$$

Calculating Running time and count of early terminated rays.

The total running time is calculated by measuring the time before the start of the process having rank 0 t_1 and and the time after the end of the process having rank 0 t_2. The difference

$$t_{total} = t_2 - t_1$$

The **time** library is used to measures time t_1, t_2.

The total count of early terminated rays is calculated in variable **early_ray_termination_count**. The variable **array** which is an array of size N {the no. of processors used} stores the count of early termination rays in each processors. Summing these values given the total count of early terminated rays.

Early Stopping

Early stopping is a concept that allows us to stop calculating values of sample points along a ray early and not computing values for all sample points for that ray. This is done to minimize computation by preventing calculations of unnecessary data points. This is meaningful as a ray will hit a complete opaque object and there is nothing behind the object. So we generally use a threshold (=1 for opacity value) to check if we can stop further computations.

Stitching and Displaying image

The local image created at each process is stitched by the process having rank 0. Once the final image is created, **PIL** library is used to render the image. The Fig. 2 shows the output of the final image. Fig. 3 is an improved version of the final image when we use log of opacity resulting in a more visually appealing image.

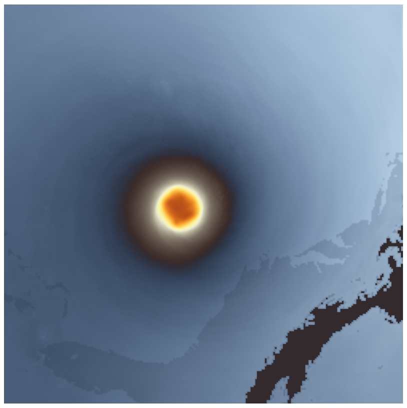

Figure 2

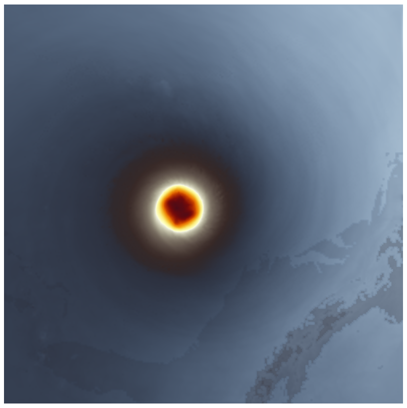

Figure 3

Comparing results

We performed the process multiple times by changing the inputs each time. The results of each run are mentioned in the following pages.

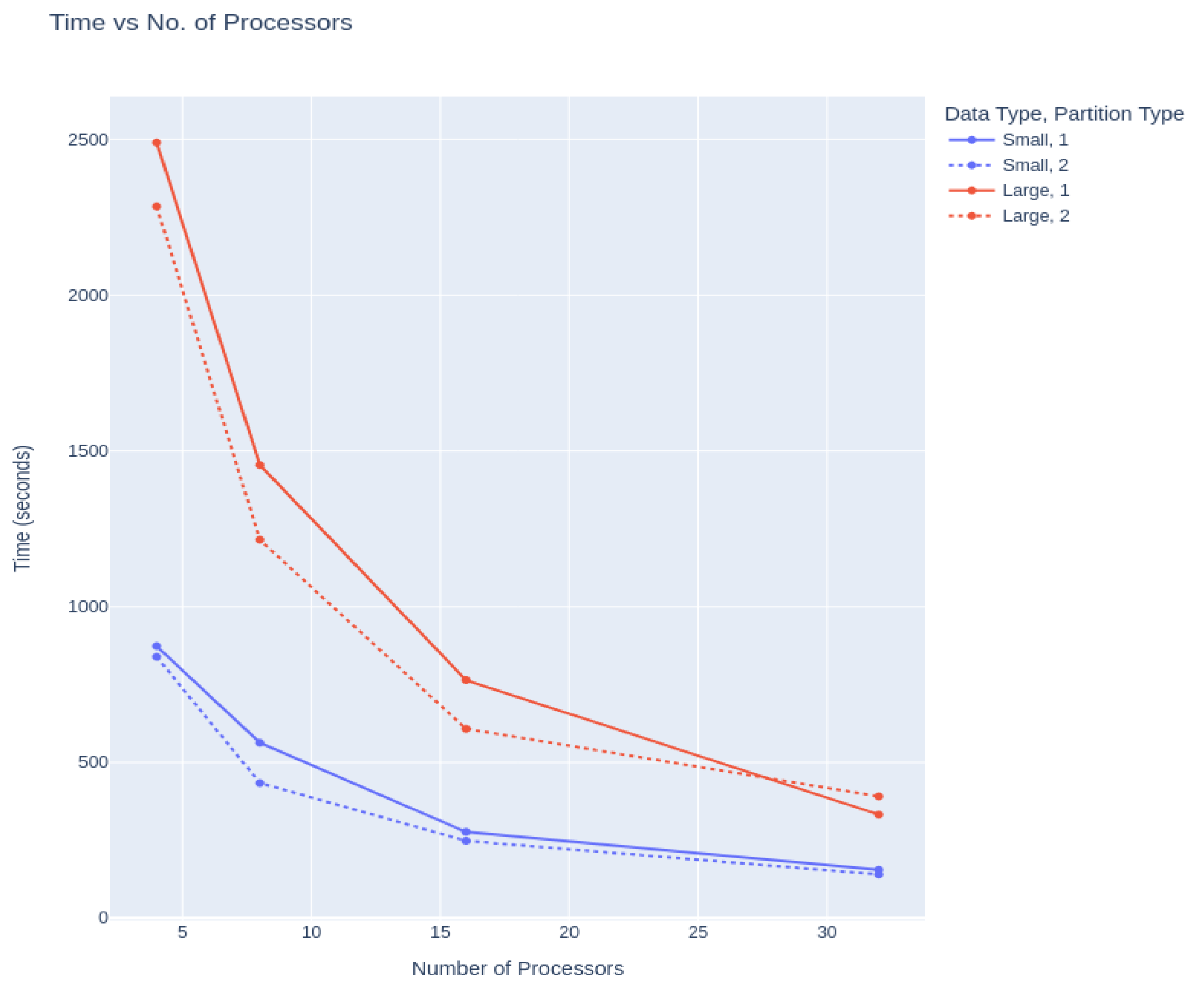

Figure 4

For Smaller Data

Changing the count of processors, Partition Type as 1

```
mpiexec -n 32 python script3.py Isabel_1000x1000x200_float32.raw 1 0.5 0 999 0 999
```

```
Extracted dimensions from filename: (1000, 1000, 200)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 999] and y: [0, 999]
Data trimmed successfully.
Partitioning data along x direction...
Data partitioned along x direction.
Stitching images.
Final image displayed.
Number of early terminated rays are: 26558
Percentage of early terminated rays are: 2.6557999999999997%
Total time taken: 155.63728213310242
```

```
mpiexec -n 16 python script3.py Isabel_1000x1000x200_float32.raw 1 0.5 0 999 0 999
```

```
Extracted dimensions from filename: (1000, 1000, 200)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 999] and y: [0, 999]
Data trimmed successfully.
Partitioning data along x direction...
Data partitioned along x direction.
Stitching images.
Final image displayed.
Number of early terminated rays are: 26558
Percentage of early terminated rays are: 2.6557999999999997%
Total time taken: 276.58596897125244
```

```
mpiexec -n 8 python script3.py Isabel_1000x1000x200_float32.raw 1 0.5 0 999 0 999
```

```
Extracted dimensions from filename: (1000, 1000, 200)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 999] and y: [0, 999]
Data trimmed successfully.
Partitioning data along x direction...
Data partitioned along x direction.
Stitching images.
Final image displayed.
Number of early terminated rays are: 26558
Percentage of early terminated rays are: 2.6557999999999997%
Total time taken: 563.1151814460754
```

```
mpiexec -n 4 python script3.py Isabel_1000x1000x200_float32.raw 1 0.5 0 999 0 999
```

```
Extracted dimensions from filename: (1000, 1000, 200)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 999] and y: [0, 999]
Data trimmed successfully.
Partitioning data along x direction...
Data partitioned along x direction.
Stitching images.
Final image displayed.
Number of early terminated rays are: 26558
Percentage of early terminated rays are: 2.6557999999999997%
Total time taken: 873.0512580871582
```

For Smaller Data

Changing the count of processors, Partition Type as 2

mpiexec -n 32 python script2.py Isabel_1000x1000x200_float32.raw 2 0.5 0 999 0 999

```
Extracted dimensions from filename: (1000,
1000, 200)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 999] and y: [0, 999]
Data trimmed successfully.
Partitioning data along both x and y
directions...
Data partitioned along both x and y directions.
Stitching images.
Final image displayed.
Number of early terminated rays are: 26558
Percentage of early terminated rays are:
2.6557999999999997%
Total time taken: 140.29941511154175
```

mpiexec -n 16 python script2.py Isabel_1000x1000x200_float32.raw 2 0.5 0 999 0 999

```
Extracted dimensions from filename: (1000,
1000, 200)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 999] and y: [0, 999]
Data trimmed successfully.
Partitioning data along both x and y
directions...
Data partitioned along both x and y directions.
Stitching images.
Final image displayed.
Number of early terminated rays are: 26558
Percentage of early terminated rays are:
2.6557999999999997%
Total time taken: 248.1450479030609
```

---

mpiexec -n 8 python script2.py Isabel_1000x1000x200_float32.raw 2 0.5 0 999 0 999

```
Extracted dimensions from filename: (1000,
1000, 200)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 999] and y: [0, 999]
Data trimmed successfully.
Partitioning data along both x and y
directions...
Data partitioned along both x and y directions.
Stitching images.
Final image displayed.
Number of early terminated rays are: 26558
Percentage of early terminated rays are:
2.6557999999999997%
Total time taken: 434.1966059207916
```

mpiexec -n 4 python script2.py Isabel_1000x1000x200_float32.raw 2 0.5 0 999 0 999

```
Extracted dimensions from filename: (1000,
1000, 200) Reading data from file...
Data read successfully.
Trimming data to x: [0, 999] and y: [0, 999]
Data trimmed successfully.
Partitioning data along both x and y
directions...
Data partitioned along both x and y directions.
Stitching images.
Final image displayed.
Number of early terminated rays are: 26558
Percentage of early terminated rays are:
2.6557999999999997
Total time taken: 838.8547790050507
```

For Larger Data

Changing the count of processors, Partition Type as 1

mpiexec -n 32 python script2.py Isabel_2000x2000x400_float32.raw 1 1.5 0 1999 0 1999

```
Extracted dimensions from filename: (2000, 2000, 400)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 1999] and y: [0, 1999]
Data trimmed successfully.
Partitioning data along x direction...
Data partitioned along x direction.
Stitching images.
Final image displayed.
Number of early terminated rays are: 101426
Percentage of early terminated rays are: 2.53565%
Total time taken: 332.86618304252625
```

---

mpiexec -n 8 python script2.py Isabel_2000x2000x400_float32.raw 1 1.5 0 1999 0 1999

```
Extracted dimensions from filename: (2000, 2000, 400)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 1999] and y: [0, 1999]
Data trimmed successfully.
Partitioning data along x direction...
Data partitioned along x direction.
Stitching images.
Final image displayed.
Number of early terminated rays are: 101426
Percentage of early terminated rays are: 2.53565%
Total time taken: 1454.9600720405579
```

mpiexec -n 16 python script2.py Isabel_2000x2000x400_float32.raw 1 1.5 0 1999 0 1999

```
Extracted dimensions from filename: (2000, 2000, 400)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 1999] and y: [0, 1999]
Data trimmed successfully.
Partitioning data along x direction...
Data partitioned along x direction.
Stitching images.
Final image displayed.
Number of early terminated rays are: 101426
Percentage of early terminated rays are: 2.53565%
Total time taken: 763.981746673584
```

---

mpiexec -n 4 python script2.py Isabel_2000x2000x400_float32.raw 1 1.5 0 1999 0 1999

```
Extracted dimensions from filename: (2000, 2000, 400)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 1999] and y: [0, 1999]
Data trimmed successfully.
Partitioning data along x direction...
Data partitioned along x direction.
Stitching images.
Final image displayed.
Number of early terminated rays are: 101426
Percentage of early terminated rays are: 2.53565%
Total time taken: 2489.8075366020203
```

For Larger Data

Changing the count of processors, Partition Type as 2

mpiexec -n 32 python script2.py Isabel_2000x2000x400_float32.raw 2 1.5 0 1999 0 1999

```
Extracted dimensions from filename: (2000, 2000, 400)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 1999] and y: [0, 1999]
Data trimmed successfully.
Partitioning data along both x and y directions...
Data partitioned along both x and y directions.
Stitching images.
Final image displayed.
Number of early terminated rays are: 101426
Percentage of early terminated rays are: 2.53565%
Total time taken: 391.2865209579468
```

mpiexec -n 16 python script2.py Isabel_2000x2000x400_float32.raw 2 1.5 0 1999 0 1999

```
Extracted dimensions from filename: (2000, 2000, 400)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 1999] and y: [0, 1999]
Data trimmed successfully.
Partitioning data along both x and y directions...
Data partitioned along both x and y directions.
Stitching images.
Final image displayed.
Number of early terminated rays are: 101426
Percentage of early terminated rays are: 2.53565%
Total time taken: 607.8282701969147
```

mpiexec -n 8 python script3.py Isabel_2000x2000x400_float32.raw 2 1.5 0 1999 0 1999

```
Extracted dimensions from filename: (2000, 2000, 400)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 1999] and y: [0, 1999]
Data trimmed successfully.
Partitioning data along both x and y directions...
Data partitioned along both x and y directions.
Stitching images.
Final image displayed.
Number of early terminated rays are: 101426
Percentage of early terminated rays are: 2.53565%
Total time taken: 1214.3682696819305
```

mpiexec -n 4 python script3.py Isabel_2000x2000x400_float32.raw 2 1.5 0 1999 0 1999

```
Extracted dimensions from filename: (2000, 2000, 400)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 1999] and y: [0, 1999]
Data trimmed successfully.
Partitioning data along both x and y directions...
Data partitioned along both x and y directions.
Stitching images.
Final image displayed.
Number of early terminated rays are: 101426
Percentage of early terminated rays are: 2.53565%
Total time taken: 2284.4824302196503
```

Sample Test Cases and Results

```
mpiexec -n 4 python script2.py
Isabel_1000x1000x200_float32.raw 1 0.75
0 999 0 999
```

```
Extracted dimensions from filename: (1000,
1000, 200)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 999] and y: [0, 999]
Data trimmed successfully.
Partitioning data along x direction...
Data partitioned along x direction.
Stitching images.
Final image displayed.
Number of early terminated rays are: 25331
Percentage of early terminated rays are:
2.5331%
Total time taken: 597.3315985202789
```

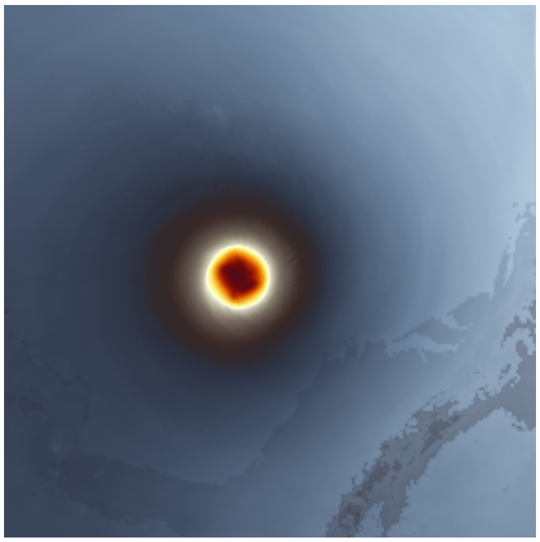
Figure 5

```
mpirun -np 8 python script2.py
Isabel_1000x1000x200_float32.raw 2 0.25
0 500 0 999
```

```
Extracted dimensions from filename: (2000,
2000, 400)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 500] and y: [0, 1999]
Data trimmed successfully.
Partitioning data along both x and y
directions...
Data partitioned along both x and y directions.
Stitching images.
Final image displayed.
Number of early terminated rays are: 16291
Percentage of early terminated rays are:
3.2516966067864272%
Total time taken: 561.8720676898956
```

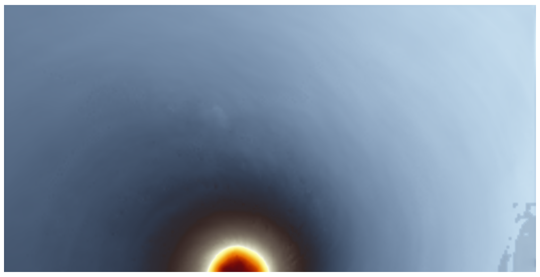
Figure 6

Sample Test Cases and Results

mpiexec -n 15 python script2.py Isabel_1000x1000x200_float32.raw 1 0.5 0 999 0 700

```
Extracted dimensions from filename: (1000, 1000, 200)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 999] and y: [0, 700]
Data trimmed successfully.
Partitioning data along x direction...
Data partitioned along x direction.
Stitching images.
Final image displayed.
Number of early terminated rays are: 26558
Percentage of early terminated rays are: 3.7885877318116976Total time taken: 179.16117548942566
```

mpiexec -n 32 python script2.py Isabel_1000x1000x200_float32.raw 2 0.35 0 500 0 999

```
Extracted dimensions from filename: (1000, 1000, 200)
Reading data from file...
Data read successfully.
Trimming data to x: [0, 500] and y: [0, 999]
Data trimmed successfully.
Partitioning data along both x and y directions...
Data partitioned along both x and y directions.
Stitching images.
Final image displayed.
Number of early terminated rays are: 16117
Percentage of early terminated rays are: 3.2169660678642713Total time taken: 102.60145211219788
```

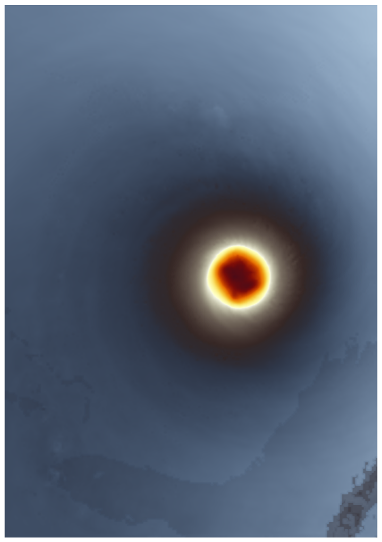
Figure 7

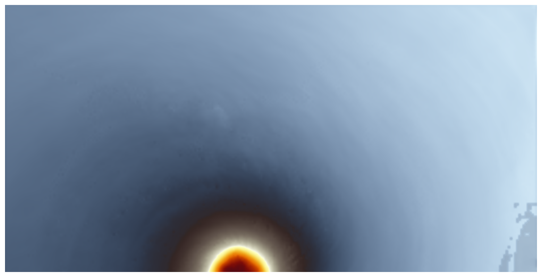
Figure 8

How to run the code?

# MPI-based Ray Casting and Visualization

The solution to this problem statement performs 3D volume rendering using ray casting. The code distributes the volume data across multiple MPI processes, applies color and opacity transfer functions, and renders the final image. It also uses early ray termination to optimize rendering.

## Features

- `MPI for Parallel Processing` - Utilizes mpi4py for distributing tasks across multiple processes.
- `Ray Casting` - Implements ray casting with color and opacity transfer functions using VTK.
- `Data Partitioning` - Supports both 1D (along x-axis) and 2D (along x and y axes) data decomposition for partitioning the 3D volume.
- `Early Ray Termination` - Optimizes rendering by terminating rays when opacity accumulates to 1.0.
- `Visualization` - Renders the final image using PIL and saves it as a PNG file.

## Requirements

To run the code, you need the following libraries:

- `mpi4py` - MPI bindings for Python, used for parallel computing.
- `numpy` - A library for numerical computations in Python, particularly for array and matrix operations.
- `vtkmodules (VTK)` - A Python wrapper for VTK, a library for 3D computer graphics, image processing, and visualization.
- `PIL (Pillow)` - A Python imaging library, used for opening, manipulating, and saving image files.
- `matplotlib` - A plotting library used for creating static, animated, and interactive visualizations in Python.
- `math` - Provides access to mathematical functions like trigonometry, logarithms, and constants.
- `sys` - Provides access to variables and functions that interact with the Python runtime environment.

- `os` - A library to interact with the operating system, used for tasks such as file manipulation and process management.
- `time` - A library for time-related functions, such as measuring execution time or working with dates and times.

Install the required libraries using:

```
pip install mpi4py numpy pillow vtk matplotlib
```

## Usage

To run the code use the following command:

```
mpiexec -n num_processes python script.py filename partition_type
step_size x_min x_max y_min y_max
```

- `<num_processes>` - Number of MPI processes to use
- `<filename>` - The name of the .raw file containing the volume data.
- `<partition_type>` - 1 for 1D partitioning (x-direction only), 2 for 2D partitioning (x and y directions)
- `<step_size>` - Step size for ray casting
- `<x_min>`, `<x_max>`, `<y_min>`, `<y_max>` - Boundaries for trimming the volume data before rendering