CS 677: Large Data Analysis and Visualisation

Group 15
Deepak Soni (241110018)
Yugul Kishor Pandey (241110083)
Kamal Kant Tripathi (241110086)

ASSIGNMENT I

A Brief Report on Code Runs and Results Obtained

Code Explanation

1 Introduction

In this assignment we have implemented a parallelized volume rendering system using MPI for distributed computation. The program processes a 3D volume data, applies ray casting algorithm and transfer function for color and opacity to generate a final rendered image.

2 Code Explanation

2.1 Functions

- **Calculate_splits(num_processes)**: Determines the optimal way to split the volume into blocks for parallel processing based on the number of processes available.
- **Divide_volume(volume_data, num_processes, partition_method, x_min, x_max, y_min, y_max)**: Divides the volume data into chunks based on the partition method:
  - **Partition method 1**: Splits volume data row wise based on the number of processes.
  - **Partition method 2**: Splits the volume into rectangular blocks based on the number of processes.

2.2 Main Function

1. **MPI Initialization**: The code starts by initializing MPI, where each process gets its rank and total number of processors.
2. **Loading Volume Data**: The root process (rank 0) loads the volume data from a file, reshapes it, and divides it among the available processes using one of the partition methods.
3. **Transfer Functions**: The root process also loads color and opacity transfer functions from files and broadcasts them to all processes.

4. **Rendering**: Each process performs ray casting on its assigned portion of the volume. The opacity and color for each voxel are interpolated using the transfer functions. Early ray termination is used to optimize performance.
5. **Gathering Results**: After rendering, each process sends its partial results back to the root process, which stitches them into the final image.
6. **Output**: The final image is saved as `Rendered_Image.png`, and performance statistics, such as the fraction of early-terminated rays and total time taken are printed.

3 Instructions to Run

3.1 Prerequisites

- Python 3
- Required Python libraries:
  - `mpi4py`
  - `numpy`
  - `PIL` (Pillow)
  - `scipy`

3.2 Running the Code

This script is designed to run in an MPI environment. To run it, we need an MPI implementation like `mpirun`.

1. Prepare the input files:
   - Volume data should be stored as a binary file (Isabel_1000x1000x200_float32.raw).
   - Transfer functions for color and opacity should be in text files named `color_tf.txt` and `opacity_tf.txt`.

2. Run the script with MPI:

```
mpirun -np <number_of_processes> python3 volume_rendering.py <volum
<partition_method> <step_size> <x_min> <x_max> <y_min> <y_max>
```

3.3 Command to run the code

```
mpirun -np <number_of_processes> python3 <filename.py> <volume_file>
<partition_method> <step_size> <x_min> <x_max> <y_min> <y_max>
```

```
An example is as under:
mpirun -np 4 python3 volume_rendering.py Isabel_1000x1000x200_float32.r
1 0.75 0 999 0 999
```

This command runs the program with 4 processes, using the first partition method and a step size of 0.75. The entire volume (from (0,0) to (999,999)) will be rendered.

4 Output

The final rendered image gets saved as `final_image.png` in the current directory. Performance statistics, including the rendering time and fraction of early-terminated rays, are printed on the console.

4.1 Test Case 1

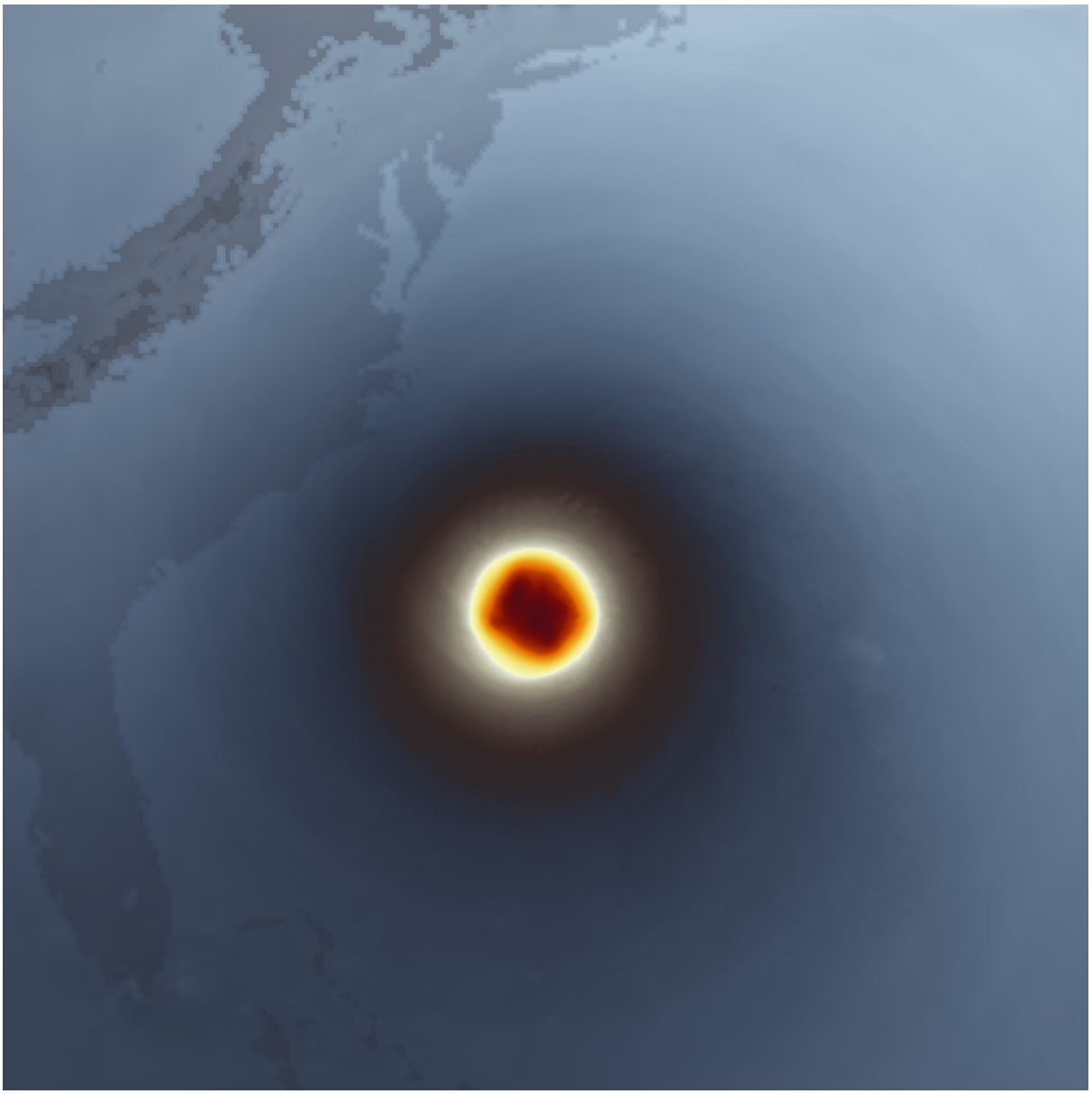
We applied volume rendering on the given volume data as under

```
(Isabel_1000x1000x200_float32.raw)
```

using partition method 1 with 4 processors and step size 0.75 with the entire volume (from (0,0) to (999,999)) to get the rendered image.
Command to run the code is as under

```
mpirun -np 4 python3 volume_rendering.py Isabel_1000x1000x200_float32.r
1 0.75 0 999 0 999
```

We got the final image as output shown as under



Time taken to generate the image is 2069.16 seconds
Fraction of ray early terminated : 100.00 %

```
yugulk24@csews2:~/Downloads/yugul$ mpirun -np 4 python3 Volume.py Isabel_1
Total rendering time: 2069.16 seconds
Fraction of rays early terminated: 100.00%
yugulk24@csews2:~/Downloads/yugul$
```

4.2 Test Case 2

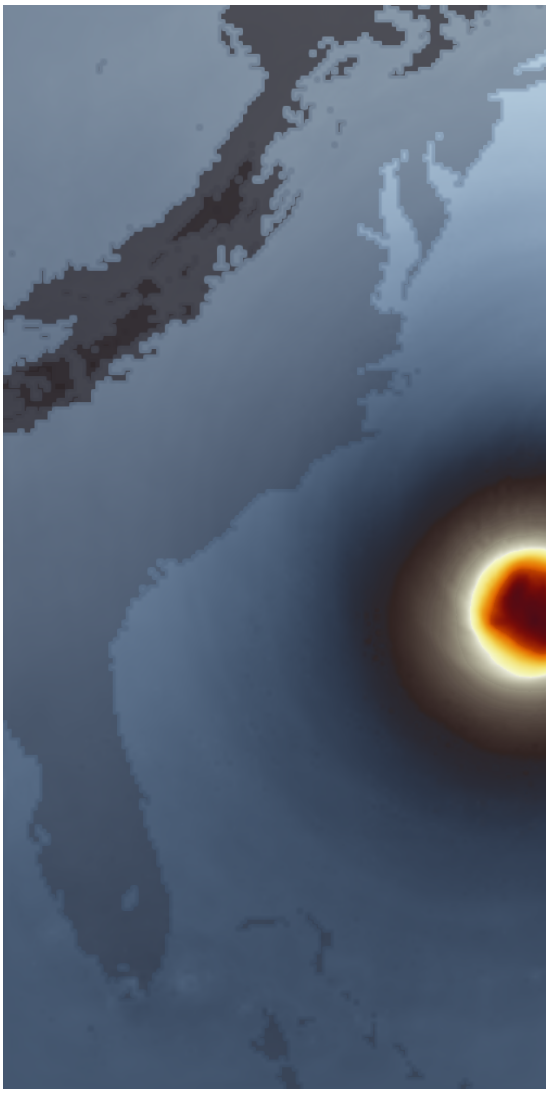
In test case 2 as well, We applied volume rendering on the given volume data as under

`(Isabel_1000x1000x200_float32.raw)`

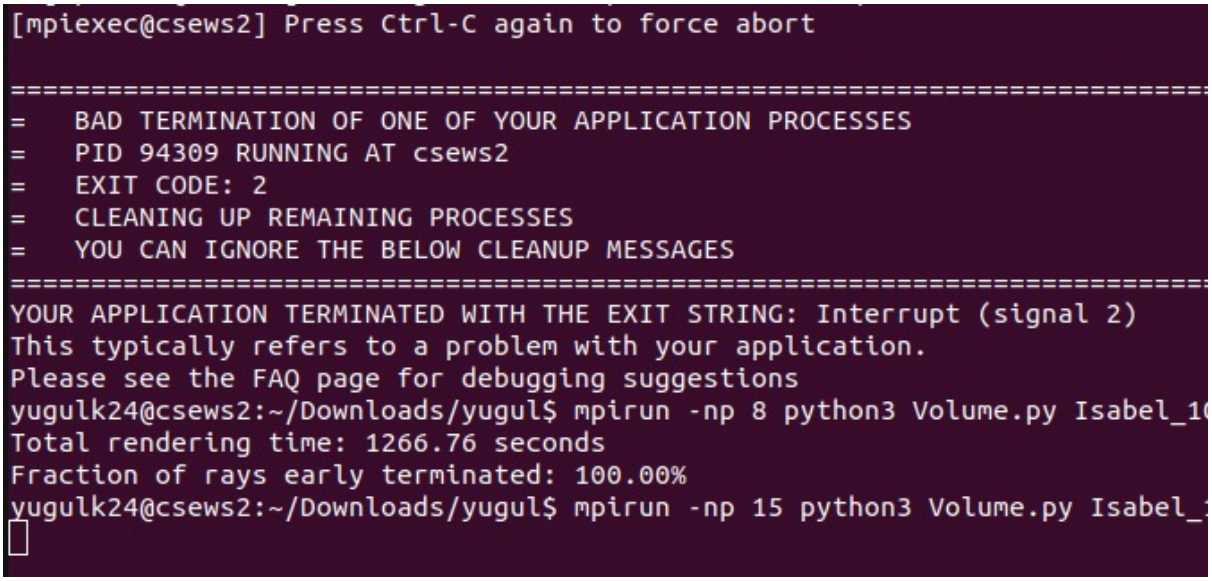
using partition method 2 with 8 processors and step size 0.25 with the entire volume (from (0,0) to (500,999)) to get the rendered image.
Command to run the code is as under

```
mpirun -np 8 python3 volume_rendering.py Isabel_1000x1000x200_float32.r
2 0.25 0 500 0 999
```

We got the final image as output shown as under



Time taken to generate the image is 1266.76 seconds
Fraction of ray early terminated : 100.00 %



4.3 Test Case 3

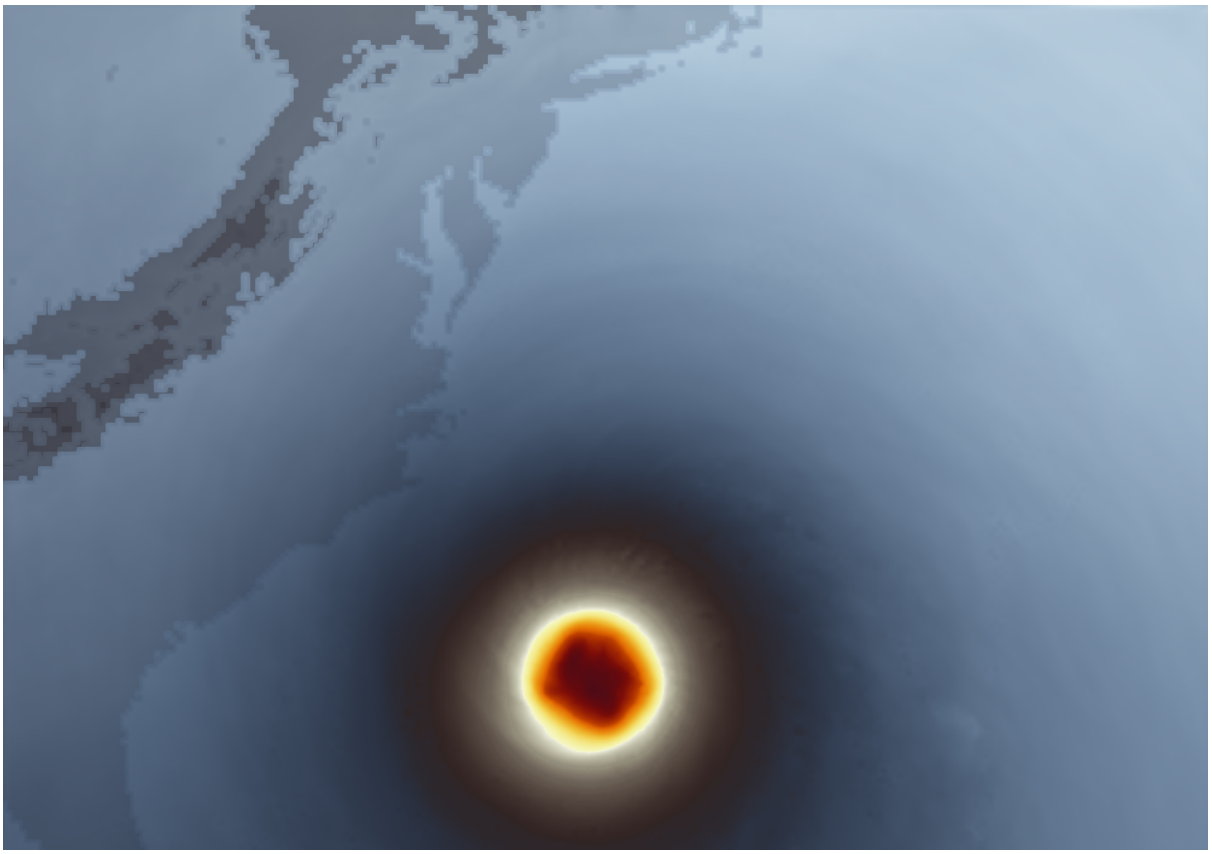
We applied volume rendering on the given volume data as under

`(Isabel_1000x1000x200_float32.raw)`

using partition method 1 with 15 processors and step size 0.5 with the entire volume (from (0,0) to (999,700)) to get the rendered image.
Command to run the code is as under

```
mpirun -np 15 python3 volume_rendering.py Isabel_1000x1000x200_float32.
1 0.5 0 999 0 700
```

We got the final image as output shown as under



Time taken to generate the image is 1078.60 seconds
Fraction of ray early terminated : 100.00 %

```
yugulk24@csews2:~/Downloads/yugul$ mpirun -np 15 python3 Volume.py Isabel_
Total rendering time: 1078.60 seconds
Fraction of rays early terminated: 100.00%
```

4.4 Test Case 4

We applied volume rendering on the given volume data as under

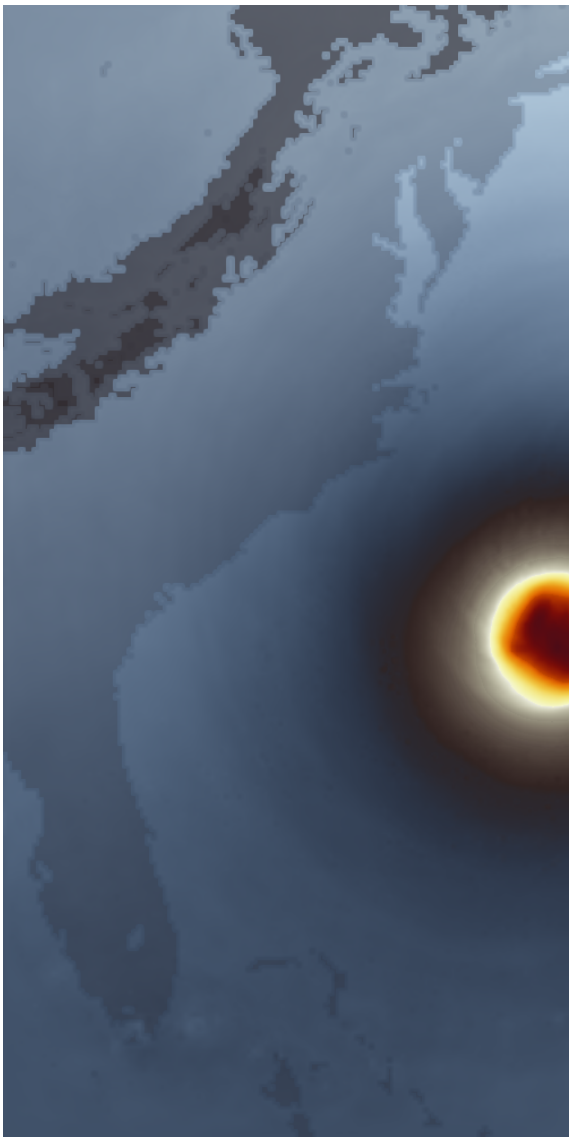
`(Isabel_1000x1000x200_float32.raw)`

using partition method 2 with 32 processors and step size 0.35 with the entire volume (from (0,0) to (500,999)) to get the rendered image.

Command to run the code is as under

```
mpirun -np 32 python3 volume_rendering.py Isabel_1000x1000x200_float32.
2 0.35 0 500 0 999
```

We got the final image as output shown as under



Time taken to generate the image is 1419.39 seconds

Fraction of ray early terminated : 100.00 %

```
Total rendering time: 1419.39 seconds
Fraction of rays early terminated: 100.00%
```