Parallel Volume Rendering Using MPI

Group 7

Saranya Pal 210930 Bhavna Jayswal 211213 Syed Adeel Ahmad 211093

Jakkampudi S K L N M Subbarayudu 210462

1 Brief Background

Volume rendering is a set of techniques used to display a 2D projection of a 3D discretely sampled data set, typically a 3D scalar field. In the field of scientific computing and visualization, volume rendering plays a pivotal role in transforming volumetric datasets into meaningful visual representations. It is extensively used in medical imaging, computational fluid dynamics, and other scientific domains.

Parallelism in computing involves the simultaneous execution of multiple tasks, enhancing computational speed and efficiency. It is particularly valuable when dealing with large datasets or complex computations. In the context of scientific visualization, parallelism is applied to techniques like ray casting, a method used for rendering realistic images from three-dimensional data.

This project tries to perform the volume rendering of 3D datasets using the ray casting algorithm and optimise the process by parallelising it by utilizing MPI.

2 Problem statement

Performing Volume Rendering of large datasets poses a challenge due to computational constraints. Traditional sequential methods fall short, especially with substantial datasets, resulting in impractical rendering times. This project addresses the need for efficient volume rendering by exploring the potential of parallel computing using MPI. The key challenge is to enhance the speed of the ray casting algorithm, a critical component in volume rendering, through parallelization across multiple processes.

3 Technical Method

We have implemented Volume Rendering using the Ray-Casting Algorithm for our project. The process begins by generating a ray for each pixel on the 2D screen of the viewer's perspective. The renderer determines the color and opacity of the pixel by considering lighting, shading, and material properties. Back-to-front compositing techniques ensure correct blending of colors and opacity, especially in the presence of overlapping objects. The accumulated color and opacity values for each pixel contribute to the final image.

Considering the computational inefficiencies of the above process, the MPI is utilized to parallelize the process. We have used the **Mpi4Py** library for the same. We start by distributing our dataset among n processes (user is free to specify the no. of processes in the beginning). Now each process will perform the ray-casting algorithm parallely. Now each process will have a fragment of the actual image, on which it has implemented the ray-casting. Finally, all of these fragements are collected from each process and merged in the source to generate the final image, the visualization of the given dataset.

Data Loading and Distribution:

- The data is loaded from a VTI file using vtkXMLImageDataReader.
- The data is distributed among 'n' MPI processes along the Z-axis.

```
a = camera_grid.shape
sizee = a
p1 = (int)(a[2] // size) * rank
p2 = (int)(((a[2] // size) * (rank + 1)) - 1)
camera_grid = camera_grid[:, :, p1:p2]
image = np.zeros((camera_grid.shape[1], camera_grid.shape
  [2], 3))
```

Listing 1: Transfer Function

Transfer Function:

- For each data slice received by an MPI process, the ray-casting algorithm is applied using the transfer function.
- The transfer function then maps scalar values to RGBA color and opacity values based on a combination of exponential functions.

$$\begin{aligned}
\text{r} &= 1.0e^{\frac{-2(x-7.5)}{1.0}} + 0.8e^{\frac{-2(x-2.0)}{1.0}} + 0.8e^{\frac{-2(x+3.0)}{0.5}} \\
\text{g} &= 0.1e^{\frac{-2(x-9.0)}{1.0}} + 0.01e^{\frac{-2(x-3.0)}{1.0}} + 0.1e^{\frac{-2(x+3.0)}{0.5}} \\
\text{b} &= 0.1e^{\frac{-2(x-9.0)}{1.0}} + 1.0e^{\frac{-2(x-3.0)}{0.1}} + 0.1e^{\frac{-2(x+3.0)}{0.5}}
\end{aligned}$$

```
root = tk.Tk()
app = PointManipulationApp(root)
root.mainloop()
opacity_values = app.get_points_values()

opacitytf = vtk.vtkPiecewiseFunction()
opacitytf.AddPoint(opacity_values[0][0], opacity_values
    [0][1])
opacitytf.AddPoint(opacity_values[1][0], opacity_values
    [1][1])
opacitytf.AddPoint(opacity_values[2][0], opacity_values
    [2][1])
```

```
opacitytf.AddPoint(opacity_values[3][0], opacity_values
    [3][1])
```

Listing 2: Transfer Function

- Now, at this moment each process will have a fragement of the actual image with it

Data Gathering and Visualization:

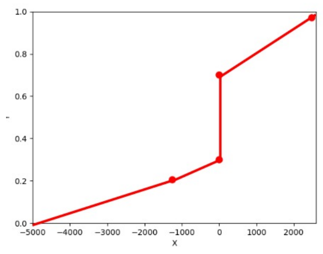
- The processed data slices are gathered to the root process (rank 0) using MPI communication.
- The final result (total_data) is visualized using Matplotlib.

```
    if rank != 0 :
        comm.Send(image, dest=0)
    else :
        total_data = image
        total_data = np.concatenate((total_data, image), 1)
        for k in range(1,size):
            comm.Recv(image, source=k)
            total_data = np.concatenate((total_data, image),
     1)
        total_data = total_data[:,p2:(p2+sizee[2]),:]
```

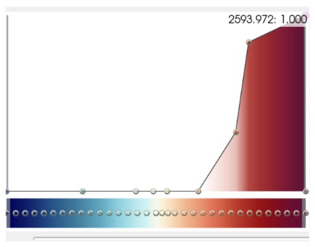
Listing 3: Transfer Function

4 Results

In the intricate tapestry of our project, we translated raw data into a visually compelling output through the application of a transfer function. The resulting image, a testament to this translation, manifested the relationships within our dataset. Seeking to elevate user interaction, we extended our efforts to construct a sophisticated Graphical User Interface (GUI). The GUI output of our implementation is as follows:-

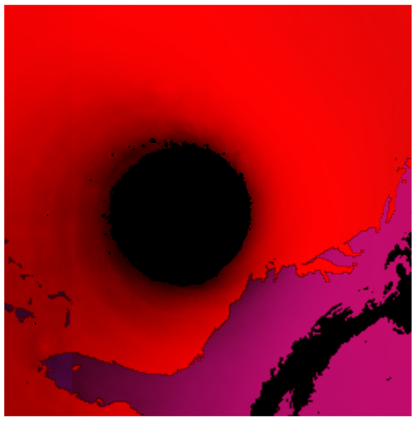


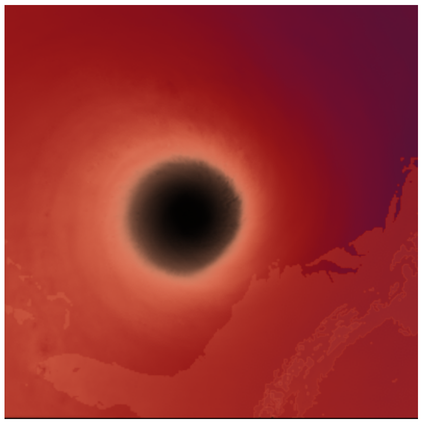
GUI from our Implementation



GUI from Paraview

This interface stands as a gateway for users to manually adjust opacity values, introducing an element of customization and control to the visualization process. All of these processes were performed parllely with the help of MPI, thus optimising the process and making it computationally feasible. In the crucible of comparison, our results stood comparable with Paraview Visualization Software, a benchmark for open-source scientific data visualization.


Output of our Implementation


Paraview Result

5 Performance Evaluation

In parallel computing, scalability refers to the ability of a system or application to efficiently utilize additional computational resources as they are added. The goal is to achieve a proportional increase in performance with an increase in the number of processors or processes. However, scalability is not always linear, and factors such as communication overhead, synchronization, and resource contention can impact performance.

In the graph, we observe an initial performance improvement up to 5 processes for the smaller-sized dataset, followed by a decline, ultimately stabilizing. Conversely, the larger-sized dataset exhibits optimal performance with 4 processes.

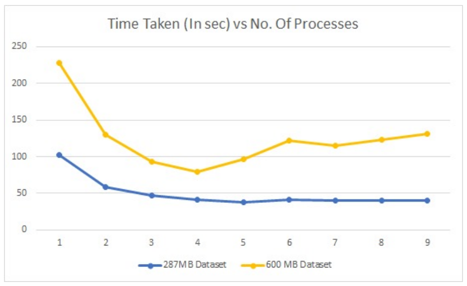


Figure 1: Time Taken vs No. of Processes for 2 Different-Sized Datasets

6 Conclusion

In conclusion, this project successfully addressed the challenge of rendering large volumetric datasets through the use of MPI for parallel computing. We achieved efficient volume rendering, enabling faster and scalable visualization of volumetric data.

The implemented parallel volume rendering technique demonstrated promising results in terms of performance improvement. The evaluation of scalability highlighted the impact of parallelization on performance, and the project's future work section suggests avenues for further enhancement. The full project can be accessed from here

7 Future Work

- **Improving the GUI:** A colour palette can be added to the GUI in order to make the data even more interactive and easier to manipulate.
- **Enhanced Transfer Functions:** Experiment with more sophisticated transfer functions for better control over color and opacity mapping. Provide user interfaces or tools for users to interactively define and adjust transfer functions.
- **Optimization for Larger Datasets:** Investigate and implement optimization techniques to handle even larger volumetric datasets efficiently. Explore parallelization strategies or algorithms that can further improve performance for extremely large datasets.

8 Work Distribution

Saranya Pal	MPI
Bhavna Jayswal	VTK
Syed Adeel Ahmad	VTK
Jakkampudi S K L N M Subbarayudu	MPI

9 References

1. https://www.youtube.com/playlist?list=PLxDvEmlm4QvgcMJLy3BiFZZ0J8fLXCuD4
2. https://examples.vtk.org/site/Python/
3. https://examples.vtk.org/site/Python/IO/ReadImageData/
4. https://www.youtube.com/watch?v=hiaHlTLN9TE
5. https://www.youtube.com/watch?v=M5WHEkbGhEE
6. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=468400&tag=
   1