CS 677 Assignment 1: Parallel Distributed Axis-aligned Volume Rendering using MPI

Introduction
This assignment aims to perform parallel volume rendering on a 3D scalar dataset using the Message Passing Interface (MPI). The rendering uses a ray casting algorithm that simulates an orthogonal projection where the view is aligned with the XY plane. The input data is a 3D scalar field, which we decompose into subdomains distributed among multiple processes. Each process handles a subdomain and performs ray casting through it, accumulating color and opacity values along the Z-axis. The results from each process are gathered and composited to form the final output image.

Methodology

1. **Input Parameters:**
   Dataset: The input 3D scalar dataset, which is specified as a file (e.g., Isabel_1000x1000x200_float32.raw).
   Partitioning: The domain can be decomposed in either 1D (along the X-dimension) or 2D (XY-dimensions).
   Stepsize: The distance between two sample points along the ray, which defaults to 0.5 but can be set to any value by the user.
   XY Bounding Box: A specified 2D bounding box within which the output image will be rendered.

2. **Domain Decomposition:**
   a. The domain is decomposed either along one dimension (1D partitioning along the X-axis) or two dimensions (2D partitioning along both the X and Y axes).
   b. For 2D partitioning, the code ensures that the number of processes in the X-direction is greater than or equal to the number of processes in the Y-direction. The number of splits in the Y-direction is taken to be the least factor of n-1 greater than 1 (1 if such factor doesn't exist).

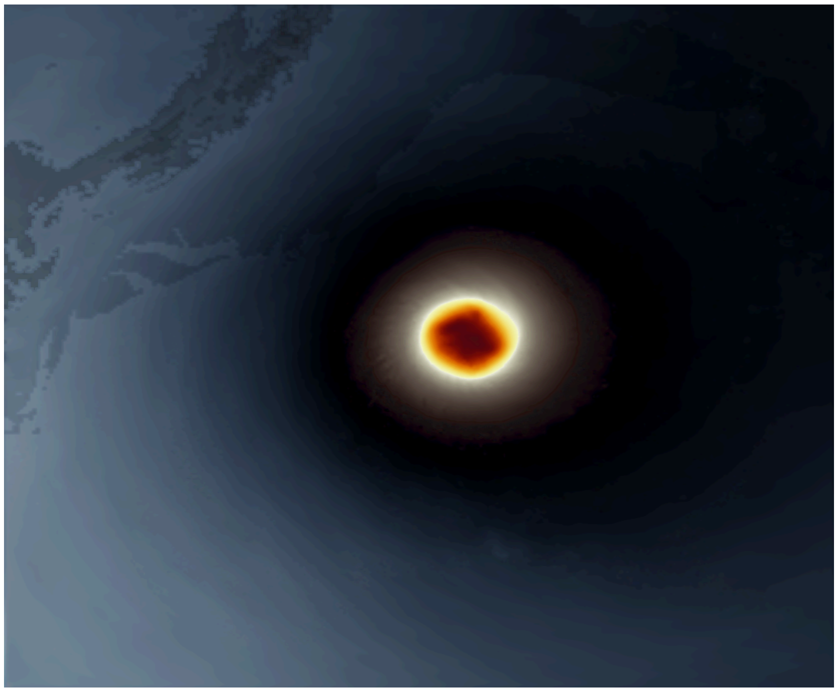
3. **Parallel Volume Rendering:**
   a. **File Reading (Rank 0):** The root process (Rank 0) reads the input data file and distributes the necessary subdomains to all other processes. Each process receives the portion of data it needs to perform ray casting within its assigned subdomain.
   b. **Ray Casting in Parallel:** Each process performs ray casting using the front-to-back compositing method. This involves casting rays along the Z-direction through the grid points in the XY plane, accumulating color and opacity values while applying the early ray termination technique to stop ray traversal when opacity reaches a certain threshold.
   c. **Data Exchange:** After processing, each process sends its partial results back to the root process (Rank 0).

    d. **Image Stitching and Output (Rank 0):** The root process gathers all partial images from the worker processes, combines them into the final output image, and saves it as a PNG file.
4. **Output:**
    a. **Rendered Image:** The final volume-rendered image of the dataset, constrained by the user-specified bounding box.
    b. **Fraction of Rays Terminated Early:** The program calculates and reports the fraction of rays that were terminated early due to the opacity threshold.
    c. **Total Execution Time:** The maximum execution time taken by any process is recorded and reported as the total execution time.

Results

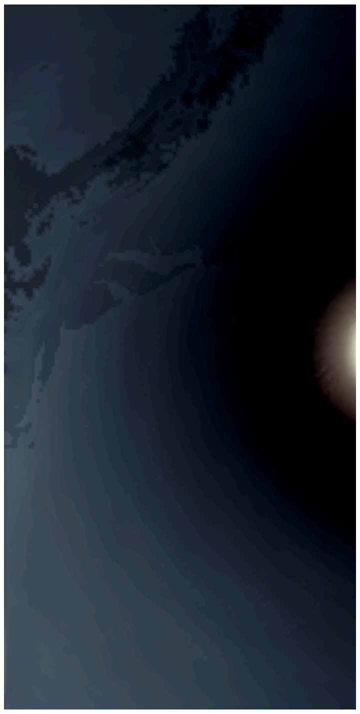
The code was tested with the following test cases:

1. **Test Case 1:**
    - Command: `mpirun -np 4 ./executable Isabel_1000x1000x200_float32.raw 1 0.75 0 999 0 1000`
    - Partitioning: 1D
    - Stepsize: 0.75
    - **Output**

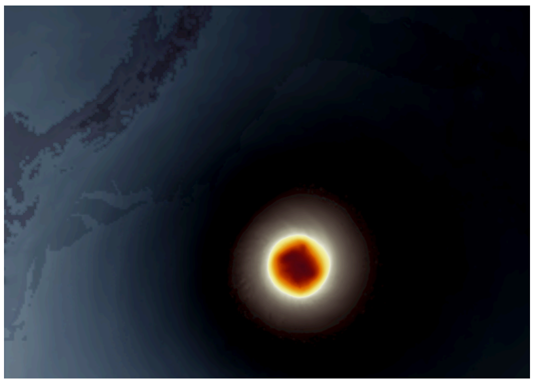


    - Total execution time: 41.555s
    - Fraction of Rays Terminated Early: 0.252
2. **Test Case 2:**

   - Command: `mpirun -np 8 ./executable Isabel_1000x1000x200_float32.raw 2 0.25 0 500 0 999`
   - Partitioning: 2D
   - Stepsize: 0.25
   - **Output**

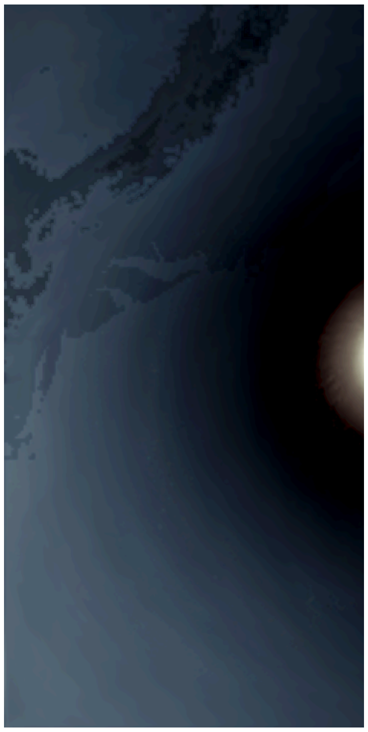


   - Total execution time: 54.154s
   - Fraction of Rays Terminated Early: 0.101
3. **Test Case 3:**
   - Command: `mpirun -np 15 ./executable Isabel_1000x1000x200_float32.raw 1 0.5 0 999 0 700`
   - Partitioning: 1D
   - Stepsize: 0.5
   - **Output**



- Total execution time: 35.028s
- Fraction of Rays Terminated Early: 0.299

4. **Test Case 4:**
   - Command: `mpirun -np 32 ./executable Isabel_1000x1000x200_float32.raw 2 0.35 0 500 0 999`
   - Partitioning: 2D
   - Stepsize: 0.35
   - **Output**



   - Total execution time: 56.709s
   - Fraction of Rays Terminated Early: 0.112

Performance Metrics

- **Fraction of Rays Terminated Early:** The program efficiently terminates rays early when sufficient opacity is reached, reducing computational load and improving performance. The fraction is computed and displayed for each test run.
- **Total Execution Time:** The time taken by each process is recorded, and the maximum time among all processes is reported as the total execution time, indicating the overall performance of the parallel rendering.

Command	Partitioning	Step size	Total execution time	Fraction of rays terminated early
`mpirun -np 4 ./executable Isabel_1000x1000x200_float32.raw 1 0.75 0 999 0 1000`	1D	0.75	41.555s	0.252
`mpirun -np 8 ./executable Isabel_1000x1000x200_float32.raw 2 0.25 0 500 0 999`	2D	0.25	54.154s	0.101
`mpirun -np 15 ./executable Isabel_1000x1000x200_float32.raw 1 0.5 0 999 0 700`	1D	0.5	35.028s	0.299
`mpirun -np 32 ./executable Isabel_1000x1000x200_float32.raw 2 0.35 0 500 0 999`	2D	0.35	56.709s	0.112

Discussion

The code implements a parallel volume rendering approach using MPI for distributed memory systems. By decomposing the domain and distributing work among multiple processes, it achieves scalability and improved performance for large datasets. The use of early ray termination further enhances efficiency by reducing unnecessary computations. The results demonstrate that the code can effectively handle different types of domain decomposition and provide the correct output based on user-defined parameters.

Conclusion

The implementation successfully achieves parallel volume rendering using MPI, with flexible input parameters for domain decomposition, step size, and bounding box specification. The output image, along with metrics such as early ray termination fraction and execution time, confirms the correctness and efficiency of the approach. Further optimization could involve load balancing and dynamic domain decomposition based on data characteristics to achieve even better performance.