CS677 : Assignment - 1

TOPICS IN LARGE DATA ANALYSIS AND VISUALIZATION

Parallel Distributed Axis-aligned Volume Rendering using MPI

GROUP 8	Divyanshu	Senthil Ganesh P	Khushwant Kaswan
	241110023	241110089	241110035

Table of Contents

Brief Overview of Ray Casting

Ray-casting is a technique that transform a limited form of data into a projection by tracing rays from the view point into the viewing volume.

Rays are cast through the volume and is sampled along equally spaced intervals. As the ray is marched through the volume, scalar values are mapped to optical properties through the use of a transfer function which results in an opacity and RGB color values for the current sample point. This color is then composited by using front-to-back or back-to-front compositing.

Front-to-back compositing

$$C'_i = C'_{i-1} + (1 - A'_{i-1})C_i A_i$$
$$A'_i = A'_{i-1} + (1 - A'_{i-1})A_i$$

C'_i is the output accumulated colour

A'_{i-1} is the accumulated alpha (opacity) value up to the previous voxel

C_i is the colour value of the current voxel

C'_{i-1} is the accumulated colour value up to the previous voxe

Brief Overview of our Task

We are given a 3D scalar dataset and we need to perform volume rendering by doing ray casting using front to back compositing method.

The domain will be decomposed in 1 dimension or 2 dimensions. Each subdomain will be handled by 1 process to perform the ray casting.

Code Sequence :

1. Dataset read by rank 0 and transformed either to 1000x1000x200 or 2000x2000x400 shape.
2. Crop/Slice the dataset according to the bounds given.
3. Rank 0 performs Domain Decomposition (1D or 2D) on cropped data and sends subdomains to all ranks
4. RayCasting on the subdomains in parallel by each rank
5. Output sub-image by each rank
6. Stitch the sub-images and generate a png file

The output of our program is:

- Volume rendered image of the bounded/cropped dataset
- fraction of the rays that are terminated early

- Time taken for data loading cropping and decomposition
- Time taken for parallel ray casting (maximum time taken by any process)

Language Used : Python

Packages/libraries used : mpi4py, numpy, matplotlib, sys, math

Implementation

Input Parameters

To handle input parameters as shown in test example, we have used Python sys module. If the number of inputs are less, we print and exit

```
    if len(sys.argv) < 8:
            print("Usage: mpirun -np <num_procs> python file_name.py
<dataset_name>
             <decomposition_type> <step_size> <x_min> <x_max> <y_min>
<y_max>")
        sys.exit()
    dataset_name = sys.argv[1]
    decomposition_type = int(sys.argv[2])
    step_size = float(sys.argv[3])
    x_min = int(sys.argv[4])
    x_max = int(sys.argv[5])
    y_min = int(sys.argv[6])
    y_max = int(sys.argv[7])
```

Shape of dataset

We are extracting the dataset shape from the dataset_name given as input and raise error when dataset name doesn't contain known dimensions.

```
def get_shape_from_dataset_name(dataset_name):
    if "1000x1000x200" in dataset_name:
        return (1000, 1000, 200)
    elif "2000x2000x400" in dataset_name:
        return (2000, 2000, 400)
    else:
        raise ValueError("Dataset name doesn't match known dimensions.")

shape = get_shape_from_dataset_name(dataset_name)
```

Reading Dataset

We have used numpy library to read the .raw file and reshaped it to the shape we got earlier.

```
def load_volume_data(filename, shape):
    data = np.fromfile(filename, dtype=np.float32).reshape(shape,
order='F')
    return data

data = load_volume_data(dataset_name, shape)
```

We took `order='F'` after checking the data point distribution in Paraview spreadsheet view.

Cropping dataset according to the bounds

Wrt to the bounds given, we crop the dataset and perform ray casting only on the new data. We are also handling if we mistakenly gave a wrong bound.

```
x_min = max(x_min, 0)
x_max = min(x_max, data.shape[1])
y_min = max(y_min, 0)
y_max = min(y_max, data.shape[0])
data = data[x_min:x_max, y_min:y_max,:]
new_shape=data.shape
```

Alternate to this approach is to always render full extent and extract the bounded image from full rendered image of data.

Bound 0 500 0 700 means 500 rows and 700 columns.

`A = A[x_min:x_max, y_min:y_max, :]` refers to slicing an array such that

```
x_min:x_max:
        x_min is the starting index for slicing rows (inclusive).
        x_max is the ending index for slicing rows (exclusive).
        This will select rows starting from x_min and going up to x_max -
1.

y_min:y_max:
        y_min is the starting index for slicing columns (inclusive).
        y_max is the ending index for slicing columns (exclusive).
        This will select columns starting from y_min and going up to y_max
- 1.

: : This is selecting all elements along the third axis.
```

Data Decomposition

```
if(decomposition_type==1):
      data =slicing_t1(size,data)
elif(decomposition_type==2):
      data =slicing_t2(size,data)
```

Type 1 Decomposition

We are splitting our dataset into slices along y axis and also handling extras.

```
def slicing_t1(num_slices, data):
  total_rows = data.shape[0]
  slice_size = total_rows // num_slices
  remainder = total_rows % num_slices
  split_indices = [slice_size * i + min(i, remainder) for i in range(1,
num_slices)]
  slices = np.split(data, split_indices, axis=0)
  return slices
```

Type 2 Decomposition

We are splitting our dataset into slices both along x and y axis and also handling cases of extras in both directions.

We have also maintained the constraint to decompose in a way such that the number of processes in x-direction is more than the number of processes in the y-direction.

```
def slicing_t2(num_slices, data):
    y_slices,x_slices = find_optimal_slices(num_slices)
    y_splits = np.array_split(data, y_slices, axis=1)
    final_splits = [np.array_split(part, x_slices, axis=0) for part in
y_splits]
    final_splits = [subarray for split in final_splits for subarray in
split]
    return final_splits
```

Data Distribution

Rank 0 assigns the first subdomain to itself and send remaining all to other ranks.

```
subdomain=data[0]
for i in range(1, size):
   comm.send(data[i], dest=i, tag=i)
```

Rank!=0 recieves the subdomains.

```
subdomain = comm.recv(source=0, tag=rank)
```

Loading color and opacity transfer functions

For colour, we are creating an array with each element of form: **(data_value,(r,g,b))**.

For opacity, we are creating an array with each element of form: **(data_value,opacity)**

```
def load_transfer_function(file_path, is_color=True):
    with open(file_path, 'r') as file:
        data = [float(val) for line in file for val in
line.strip().replace(',', '').split()]
    if is_color:
        return [(data[i], tuple(data[i+1:i+4])) for i in range(0,
len(data), 4)]
    else:
        return [(data[i], data[i+1]) for i in range(0, len(data), 2)]
```

Raycasting with Front-to-back compositing

- FOR each pixel position (x, y) in the image, we are traversing through the z values incrementing it with step_size.
- We interploate the data value at (x,y,z+step_size) using values at data values index int(z) and int(z)+1 while being in depth bounds.
- Using this interploated value, we interploate color and opacity at that point.
- Use front to back compositing.
- For early ray termination, we have chosen opacity threshold of 0.98.

```
def ray_casting(subdomain, opacity_points, color_points, step_size,rank):
    height, width, depth = subdomain.shape
    img = np.zeros((height, width,3))
    terminated_rays = 0

    for y in range(width):
        for x in range(height):
            accumulated_color = np.zeros(3)
            accumulated_opacity = 0
            z = 0.0
            while z < depth:
                data_val = interpolate_value(subdomain,x,y,z)
                color = np.array(interpolate_color(data_val, color_points))
                opacity = interpolate_opacity(data_val, opacity_points)

                # Front to back compositing logic
```

```
                if accumulated_opacity >= 0.98:
                    terminated_rays += 1
                    break

                z += step_size

            img[x, y,:] = accumulated_color
    return img,terminated_rays
```

Stitching Gathered Images

We split the final images according to the method they were decomposed using **hstack** and **vstack** available in numpy.

```
gathered_images = comm.gather(img, root=0)

if rank == 0:
  if decomposition_type == 1:
          final_image
            # Stiching Logic for Type 1 Decomposition
        else:
          final_image
            # Stiching Logic for Type 2 Decomposition

  file_name=f"bounded_{size}_{decomposition_type}_{step_size}.png"
  plt.imsave(file_name,final_image)
```

Fraction of the rays that are terminated early

Used MPI.reduce to calculate sum of all terminated_rays count.

```
rays_terminated_early_total = comm.reduce(terminated_rays, op=MPI.SUM,
root=0)
if rank == 0:
  total_rays = new_shape[0] * new_shape[1]
  fraction_early_terminated = rays_terminated_early_total / total_rays
```

Max time taken by any rank for raycasting

We implemented different methods using time(), MPI.Wtime() giving same max value.

```
times2=[]
times3=[]
start_time3,start_time2 = time.time(),MPI.Wtime();
img, terminated_rays = ray_casting(subdomain,opacity_points, color_points,
```

```
step_size,rank)
end_time3,end_time2 = time.time(),MPI.Wtime();
time_taken2=end_time2-start_time2
time_taken3=end_time3-start_time3
times2.append(time_taken2)
times3.append(time_taken3)
```

```
if rank == 0:
        print(f"Maximum time taken 2 by any rank: {max(times2)} seconds")
#M1
        print(f"Maximum time taken 3 by any rank: {max(times3)} seconds")
#M2
        print(f"Time taken for parallel ray casting : {end_time2 -
start_time2}sec or
                {((end_time2 - start_time2)/60):.4f}  min") #M3
```

```
~/Desktop/LDAV  took 28s
>>> mpirun -np 4 python test.py  Isabel_1000x1000x200_float32.raw 2 50 0 1999 0 1999
Time taken for data loading and decomposition according to type 2 : 2.7626sec
Maximum time taken 2 by any rank: 17.668244333 seconds
Maximum time taken 3 by any rank: 17.66825819015503 seconds
Time taken for parallel ray casting : 17.668244333sec or 0.2945  min
```

We have used M3 for our Results.

Results

1000x1000x200 dataset

-np	Step Size	Type	Bounds	t1 (sec)	t2 (sec)	t2 (min)	ETF
4	0.75	1	[0-999][0-1000]	0.12	885.36	14.75	0.78
8	0.25	2	[0-500][0-999]	0.13	253.19	4.22	1
15	0.5	1	[0-999][0-700]	0.14	293.20	4.88	1
32	0.35	2	[0-500][0-999]	0.46	191.77	3.19	1

- Type: Domain Decomposion type 1D or 2D
- t1: Time taken for data loading and decomposition in seconds
- t2: Maximum time taken by any rank in seconds
- ETF: Fraction of rays terminated early

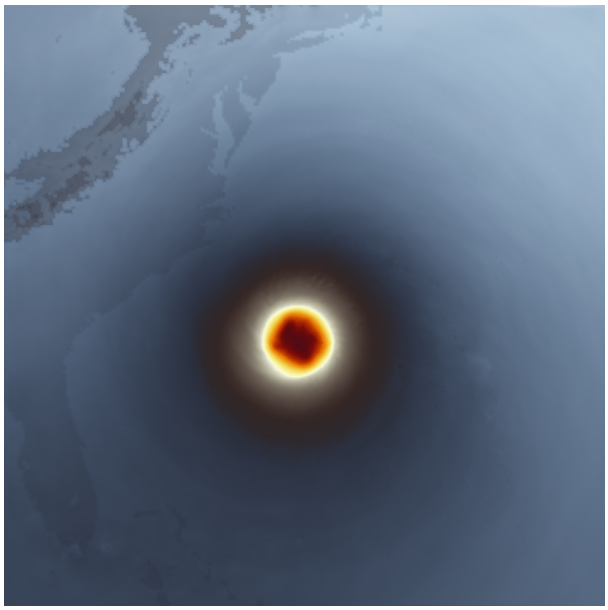
```
khushwantk24@csews5:~/Downloads/DV$ mpirun -np 32 python3 test.py Isabel_1000x1000x200_float32.raw 2 0.35 0 500 0 999
Time taken for data loading and decomposition according to type 2 : 0.4661sec
Time taken for parallel ray casting : 191.7756sec or 3.1963  min
Fraction of rays terminated early: 499500 / 499500= 1.0000
khushwantk24@csews5:~/Downloads/DV$ mpirun -np 15 python3 test.py Isabel_1000x1000x200_float32.raw 1 0.5 0 999 0 700
Time taken for data loading and decomposition according to type 1 : 0.2132sec
Time taken for parallel ray casting : 293.2003sec or 4.8867  min
Fraction of rays terminated early: 699300 / 699300= 1.0000
khushwantk24@csews5:~/Downloads/DV$ mpirun -np 8 python3 test.py Isabel_1000x1000x200_float32.raw 2 0.25 0 500 0 999
Time taken for data loading and decomposition according to type 2 : 0.1396sec
Time taken for parallel ray casting : 253.1975sec or 4.2200  min
Fraction of rays terminated early: 499500 / 499500= 1.0000
khushwantk24@csews5:~/Downloads/DV$ mpirun -np 4 python3 test.py Isabel_1000x1000x200_float32.raw 1 0.75 0 999 0 1000
Time taken for data loading and decomposition according to type 1 : 0.1304sec
Time taken for parallel ray casting : 885.3642sec or 14.7561  min
Fraction of rays terminated early: 787246 / 999000= 0.7880
khushwantk24@csews5:~/Downloads/DV$
```

In the follwing results, the Full Extent Rendered Image has been generated seperately.

The times shown are only for the bounded image, not for the full extent image.

1. **mpirun -np 4 python file.py Isabel_1000x1000x200_float32.raw 1 0.75 0 999 0 1000**

```
khushwantk24@csews5:~/Downloads/DV$ mpirun -np 4 python3 test.py Isabel_1000x1000x200_float32.raw 1 0.75 0 999 0 1000
Time taken for data loading and decomposition according to type 1 : 0.1304sec
Time taken for parallel ray casting : 885.3642sec or 14.7561  min
Fraction of rays terminated early: 787246 / 999000= 0.7880
khushwantk24@csews5:~/Downloads/DV$
```



Full Extent Rendered Image

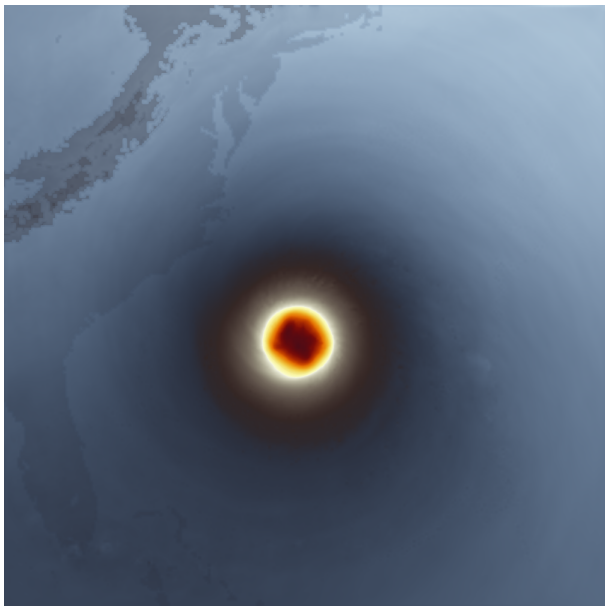
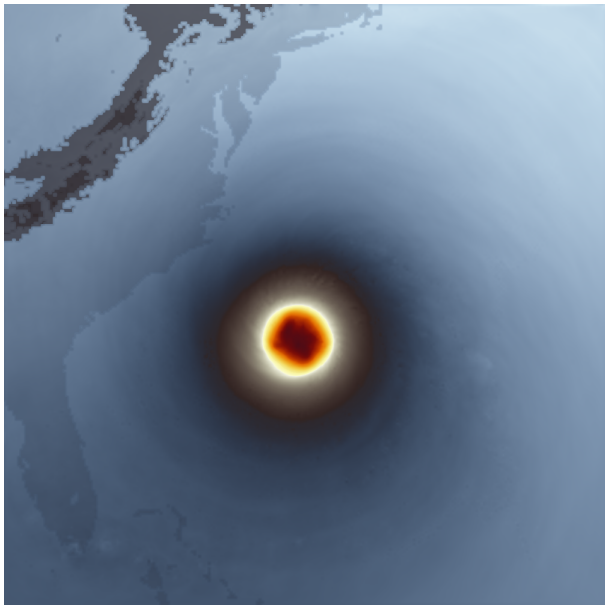


Image within the bounding box

2. **mpirun -np 8 .python file.py Isabel_1000x1000x200_float32.raw 2 0.25 0 500 0 999**

```
khushwantk24@csews5:~/Downloads/DV$ mpirun -np 8 python3 test.py Isabel_1000x1000x200_float32.raw 2 0.25 0 500 0 999
Time taken for data loading and decomposition according to type 2 : 0.1396sec
Time taken for parallel ray casting : 253.1975sec or 4.2200  min
Fraction of rays terminated early: 499500 / 499500= 1.0000
khushwantk24@csews5:~/Downloads/DV$
```



Full Extent Rendered Image

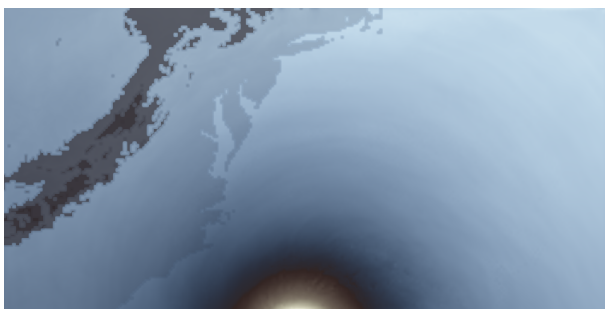
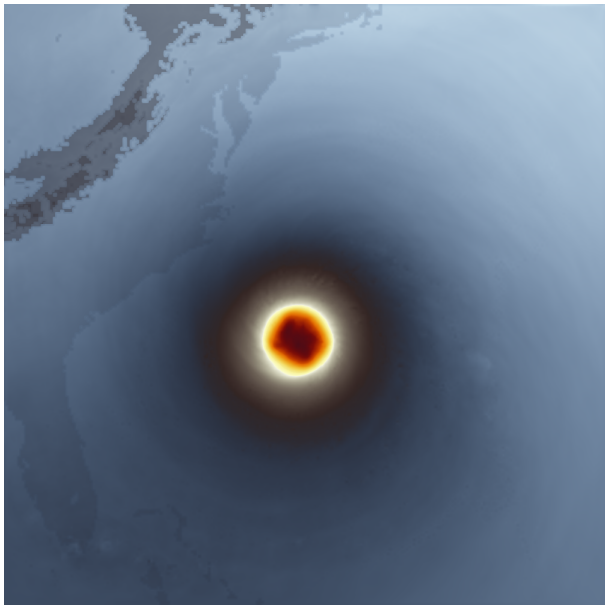


Image within the bounding box

3. **mpirun -np 15 python file.py Isabel_1000x1000x200_float32.raw 1 0.5 0 999 0 700**

```
khushwantk24@csews5:~/Downloads/DV$ mpirun -np 15 python3 test.py Isabel_1000x1000x200_float32.raw 1 0.5 0 999 0 700
Time taken for data loading and decomposition according to type 1 : 0.2132sec
Time taken for parallel ray casting : 293.2003sec or 4.8867  min
Fraction of rays terminated early: 699300 / 699300= 1.0000
khushwantk24@csews5:~/Downloads/DV$
```



Full Extent Rendered Image

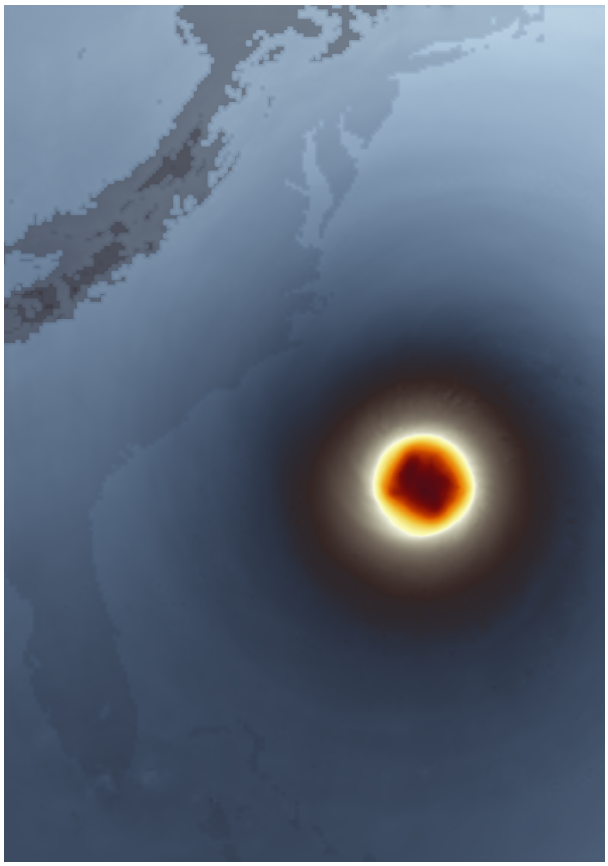
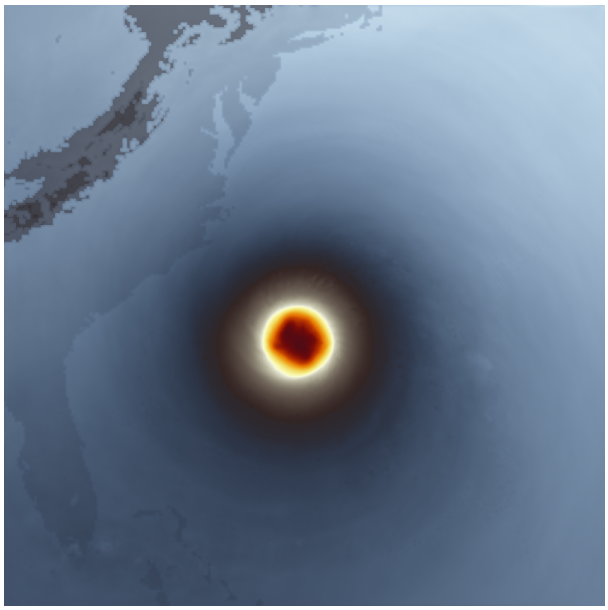


Image within the bounding box

4. **mpirun -np 32 python file.py Isabel_1000x1000x200_float32.raw 2 0.35 0 500 0 999**

```
khushwantk24@csews5:~/Downloads/DV$ mpirun -np 32 python3 test.py Isabel_1000x1000x200_float32.raw 2 0.35 0 500 0 999
Time taken for data loading and decomposition according to type 2 : 0.4661sec
Time taken for parallel ray casting : 191.7756sec or 3.1963  min
Fraction of rays terminated early: 499500 / 499500= 1.0000
khushwantk24@csews5:~/Downloads/DV$
```



Full Extent Rendered Image

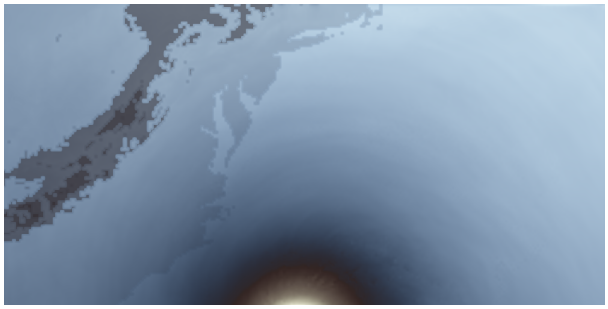


Image within the bounding box

Sample testcase on 2000x2000x400 dataset

-np	Step Size	Type	Bounds	t1(sec)	t2(sec)	t2(min)	ETF
15	0.5	1	[0-1999][0-700]	30.47	435.42	7.25	1
32	0.35	2	[0-500][0-1999]	26.78s	348.10	5.80	1

- Type: Domain Decomposion type 1D or 2D
- t1: Time taken for data loading and decomposition in seconds
- t2: Maximum time taken by any rank in seconds
- ETF: Fraction of rays terminated early

```
khushwantk24@csews5:~/Downloads/DV$ mpirun -np 32 python3 test.py Isabel_2000x2000x400_float32.raw 2 0.35 0 500 0 1999
Time taken for data loading and decomposition according to type 2 : 26.7802sec
Time taken for parallel ray casting : 348.1052sec or 5.8018  min
Fraction of rays terminated early: 999500 / 999500= 1.0000
khushwantk24@csews5:~/Downloads/DV$ mpirun -np 15 python3 test.py Isabel_2000x2000x400_float32.raw 1 0.5 0 1999 0 700
Time taken for data loading and decomposition according to type 1 : 30.4719sec
Time taken for parallel ray casting : 435.4244sec or 7.2571  min
Fraction of rays terminated early: 1399300 / 1399300= 1.0000
khushwantk24@csews5:~/Downloads/DV$
```

1.**mpirun -np 15 python file.py Isabel_1000x1000x200_float32.raw 1 0.5 0 1999 0 700**

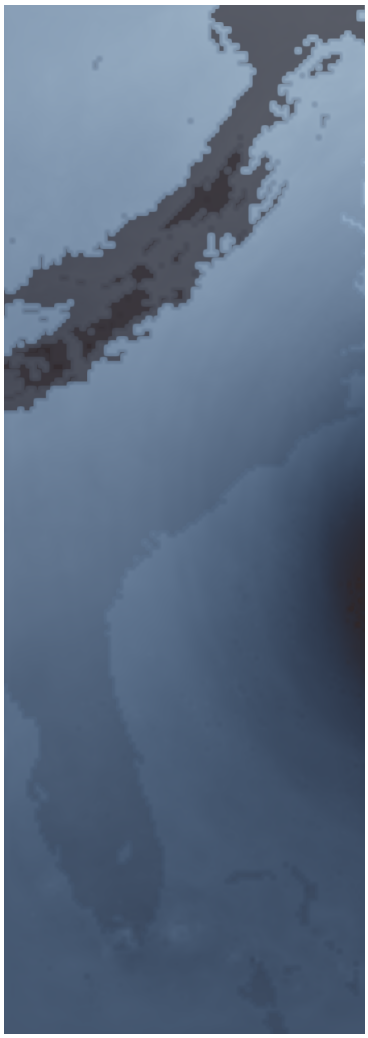


Image within the bounding box

2.**mpirun -np 32 python file.py Isabel_1000x1000x200_float32.raw 2 0.35 0 500 0 1999**

Image within the bounding box

Possible Caveats

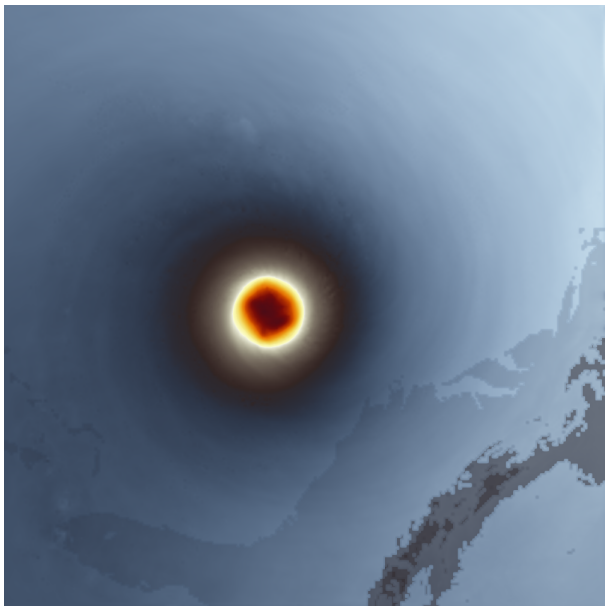
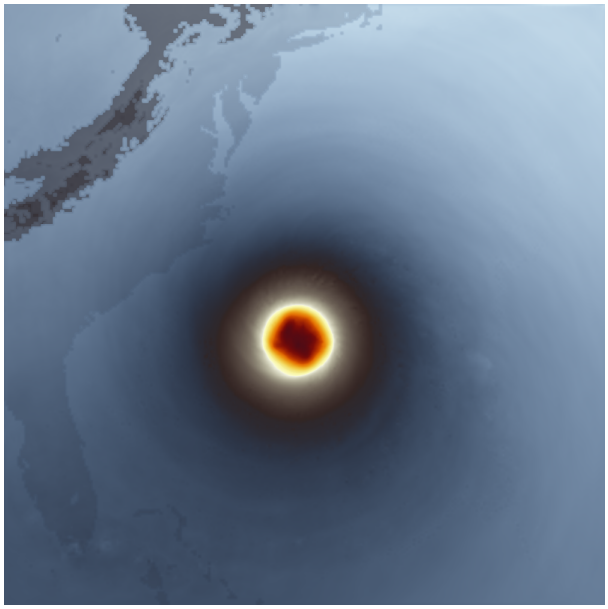


Image ParaView Gives after Clicking +Z (Looking down x axis from (0,0,1))



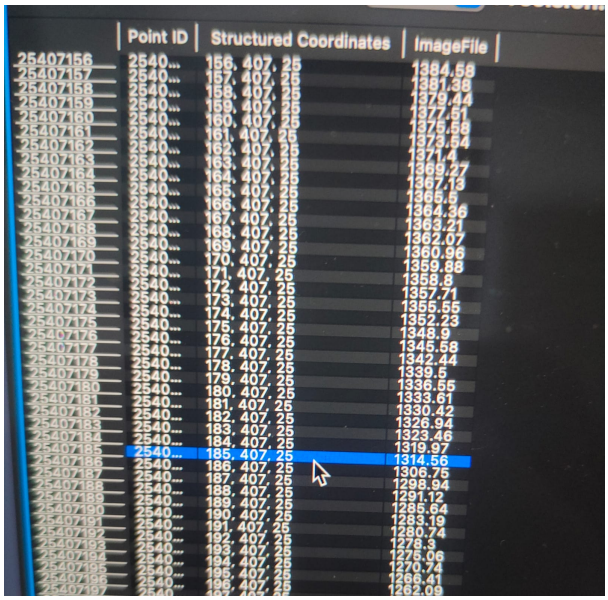
Our Full Extent Rendered Image

We can get same output by flipping and rotaing our full extent rendered image 90deg

```
flipped_img = np.flipud(final_image)
rotated_img = np.rot90(flipped_img)
plt.imsave(file_name,rotated_img)
```

But we have verified our main data distribution with paraView spreadsheet view and its the same as our data distribution.

Our 3D array



ParaView spreadsheet View

How to run our code

We have used mpi4py, numpy and matplotlib libraries.

To install we have created a install_libraries.sh

If that doesn't work :

```
pip install mpi4py
pip install numpy
pip install matplotlib
```

To run our code,

1. Place the dataset,color_Tf.txt,opacity_TF.txt in the same folder as code.py
2. Use the following command to run our code :

   ```
   mpirun -np 4 python3 code.py Isabel_1000x1000x200_float32.raw 1 0.75 0 999 0 1000
   ```

   ```
   mpirun -np 8 python3 code.py Isabel_1000x1000x200_float32.raw 2 0.25 0 500 0 999
   ```

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

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

Use python or python3 according to what is installed.

Output will be a file named `bounded_{np}_{type}_{step_size}`

Images used in the report are reduced to their 50% height and width.

Original render Images are available in images folder.

Folder (images/1/) for the test examples.

Folder (images/2/) for the our sample run on bigger dataset.

Folder (images/full/) contains the full data extent render.