How to Use the ExMage Library

1 Introduction

The ExMage library is intended to make explorable images easy to create. This document will show how to use it in a simple simulation. For simplicity, the simulation code is omitted in favor of simply loading precalculated timesteps from a file. However, interaction with the ExMage library is identical when the particle positions and properties are calculated from a real simulation. All code and resources for this example project are available at the ExMage project site.

2 Example API Use

2.1 Setup

The ExMage library is intended to be used within an MPI executable. After the MPI setup, each node is expected to create an Explorable object with a global configuration as input, as follows:

```
Explorable explorable;
explorable.setConfigFile("configure.json"); // If not set, it's default to be configure.json
```

2.2 Generation

```
for (int timestep = range[0]; timestep <= range[1]; ++timestep)
{
    //.
    //... Field generation...
    //.
    explorable.update(fields);
}
```

2.3 Output

After generation of the explorable image has completed, the various nodes must combine their outputs into a single output image. This is performed with the following code:

```
explorable.output();
```

2.4 Complete Listing

```
#include <iostream>
#include <fstream>
#include <cassert>
#include <cstdlib>
#include <ctime>
#include <string>

#include "mpi.h"
#include "VectorFieldReader.h"
#include "Explorable.h"

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);
    time_t start, end;
    double dif;
    time(&start);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // read vector field
    VectorFieldReader reader;
    std::vector<float*> fields;
    assert(reader.read());
    fields = reader.getFields();

    //
    //
    // This is the expected initialization code.
    // 1. Initialize our class
    // 2. Set properties that should be obtained from the simulation codes
    // 3. Otherwise, the properties are set by the config file
    //
    //
    Explorable explorable;
    explorable.setConfigFile("configure.json"); // If not set, it's default to be configure.json

    //
    //
    // Here is the fake Simulation main loop
    // Currently, I am still reading the timestep from the config file.
    // Later on, I should remove that because the simulation should be taking care of that.
    //
    //
    ConfigReader& config = ConfigReader::getInstance();
    std::vector<int> range = config.get("input.time").asArray<double, int>();
    for (int timestep = range[0]; timestep <= range[1]; ++timestep)
    {
        //
        //
        // Inside the Simulate loop.
        //
        // The first 3 fields are the velocity fields, the 4th/last field is
        // the scalar field that we do visualization on.
        //
        explorable.update(fields);
    }

    //
    //
    // After the Simulation is done, we need the simulation to initiate the output command.
    //
    //
    explorable.output();

    time(&end);
    dif = difftime(end, start);
    MPI_Finalize();
    return 0;
}
```

3 Configuration

The configuration object is a simple JSON object with several specific configuration parameters. Most of these are self-explanatory. Specifically, the necessary configuration paramters for the camera are its position, focal point and up vector. Each light is specified by its position. The output is described by the output image width and height, and its filename. The MPI domain is expected to be evenly divided among nodes in three dimensions. Currently, explorable images are expected to be generated solely from points and tube-segments, the attributes of which may also be specified. An example of all of these configurations is shown below.

```
{
  "camera": {
    "position": [75.0, 1075.0, 75.0],
    "focal": [75.0, 75.0, 75.0],
    "up": [1.0, 0.0, 0.0]
  },
  "light": {
    "position": [150.0, 150.0, 75.0]
  },
  "output": {
    "resolution": [512, 512],
    "file": "supernova"
  },
  "tube": {
    "count": 15,
    "radius": 0.1,
    "velocity": 2.0,
    "gap": 10.0
  },
  "domain": {
    "count": [2, 2, 2],
    "volume": [75, 75, 75]
  }
}
```