Modeling the Flow of the Interstellar Medium Within Localized Sectors of Space

New Mexico
Supercomputing Challenge
Final Report
April 4, 2012

Team Number 4
AIMS@UNM

Team Members

Louis Jencka

Randall van Why

Nico Ponder

Stefan Klosterman

Jake Kileen

Teacher

Mr. Harris

Table of Contents

1. Executive Summary

The main goal of our project was to model and optimize the flow of the interstellar medium. Our program was designed to accomplish the following core functions: The program must accurately represent the interstellar medium, must accurately and precisely model the interactions between the particles in the interstellar medium on a macroscopic scale, and must be optimized such that it can run on a budget desktop computer or a laptop. Firstly, we built a display mechanism that integrates OpenGL with C++, and ensured that it was capable of building and running across multiple platforms using CMake. We then implemented a spherical coordinate system into this, allowing us to use cubes to represent sections of the interstellar medium.

Primary focuses of our project included modularity, accessibility, and open-development. We were able to make our program work across multiple platforms, made our project accessible throughout and after its development using GitHub, and maintained its efficiency. We created a system which allowed for creation of on-the-fly scenarios in our model, allowing users to adapt our model to their needs (see Appendix B).

Our methods of implementation included a Mathematical model which computed the various scenarios. The Mathematical model then fed the data to the graphical OpenGL model. The model proved to be a fairly accurate representation of the gravitation interaction in the Interstellar Medium.

The results are consistent with our original projections. Our model conducts calculations and computations more accurately and efficiently than standard N-

Body simulations. The code, at its current state is capable of simulating the movement of diffuse areas of matter. The code can and will be adapted in a possible future project to incorporate charge and thermodynamics.

2. Problem Statement

The interstellar medium contains gases, interstellar dust, and nebulae. Modeling the formation of nebulae and other formations in the interstellar medium could provide insight into the formation of larger structures.

How can we effectively model the formation of structures in the interstellar medium while maintaining efficiency and modularity?

3. Introduction

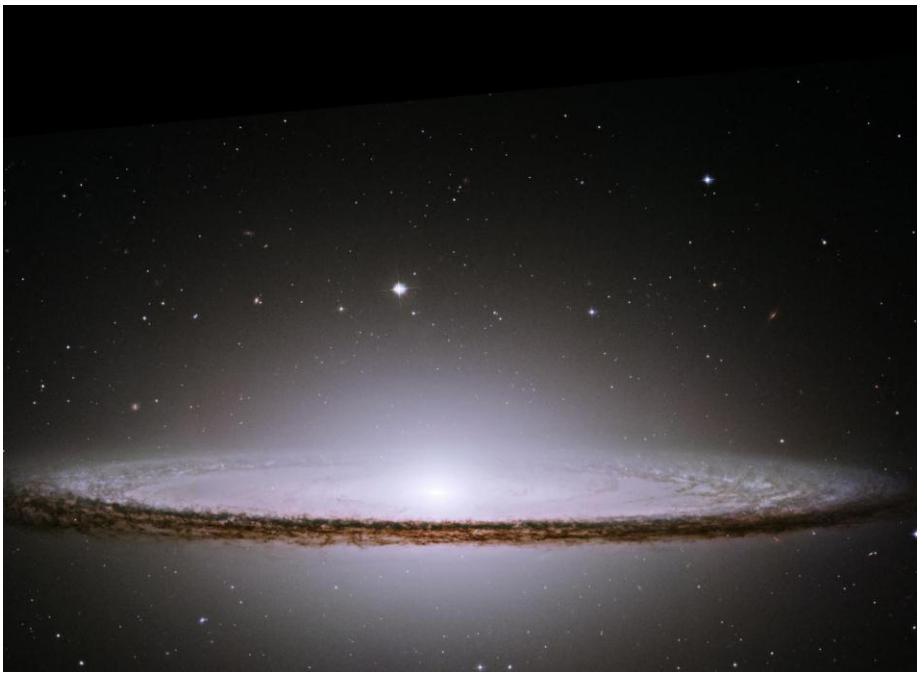
3.1 What is the interstellar medium?

The interstellar medium is a collection of gases and interstellar dust in the low-density regions of space that lies between stars. The interstellar medium is composed mainly of gases: about 99% of the material found in the interstellar medium is a gas. Approximately 75% of the gas in the interstellar medium is hydrogen, and around 25% is helium. The remaining 1% of the matter in the interstellar medium is interstellar dust in the form of silicates, carbon, ice, or iron compounds. Most regions of the interstellar medium are vast and empty. The density of these regions is about 1 atom per cubic centimeter. When compared to the density of air on Earth, which contains 3.0×10^{19} molecules per cubic centimeter, one can see just how empty the interstellar medium is. (University of New Hampshire Experimental Space Plasma Group, n.d.) However, in regions of the interstellar medium such as nebulae, conditions are much different.

3.2 What are nebulae?

Nebulae are hotbeds of activity in the interstellar medium. They are denser than the other, emptier regions of the interstellar medium (Planetary Nebulae, n.d.) and can reach high temperatures. Nebulae that met the right conditions are capable of forming new stars, or even solar systems, over time. There are five kinds of nebulae: Emission nebulae, reflection nebulae, dark nebulae, planetary nebulae, and supernova remnants.

Emission nebulae are gathered clouds of high temperature gas, typically hydrogen. Emission nebulae exist relatively near stars, as the energy the stars emit causes the atoms in emission nebulae to emit light as they change energy states. Reflection nebulae are simply collections of dust and gas that reflect light. This light comes from nearby sources of light, and appears blue. Dark nebulae are clouds of dust that keep light from objects behind them from fully showing. Planetary nebulae the result of dying stars. They are essentially hollow spheres of gas that are illuminated by the dying star at their center. Supernova remnants are byproducts of the explosive endings of stars. The supernova expels matter into space, where it is then illuminated by other remnants of the star. This cloud of matter glows with the remains of the star that created it. (Knight, n.d.)



The Sombrero Nebula
(source: www.point64.com)

4. Method

The team has developed a hybrid statistical/graphical model. The model is written in the C++ Programming language and incorporates the OpenGL, SDL, BOOST, AGAR, and libXML2 libraries. The code consists of a graphical representation (SDL & OPENGL) wrapped in a rendered Graphical User Interface (AGAR) that incorporates both modularity (libXML) and multi-threading (BOOST).

The Code (Overview)

The basic code was broken up into four core concepts:

- *The Cube*- An area of space whose contents are to be treated as a single mass created from a composite of the particles within it.
- *The Conglomerates*- A larger version of the Cube which consists of multiple Cubes. It, like its parts, makes a whole from its sums.
- *The Core*- The section of the code that processes the data, and manipulates it according to certain rules
- *The Provehamus*- A graphical representation of the data being acted upon, as well as an interface to manipulate the rest of the program.

An Elaboration

The Cube

Each data cube consists of only a few distinct pieces, and will chiefly serve to hold these components:

Specific Component Masses

Average density, as well as total Mass

The Cube will be treated as a mass of uniform properties, and movement of mass should be facilitated using the cubes. Each cube acts as a body that is affected by the other bodies in space.

The Conglomerate

The conglomerate functions as an array of cubes added up together. The conglomerate is the team's effort to implement optimization into the code[1]. Data cubes that are not being manipulated or observed are devolved into a conglomerate. The conglomerate averages up all the components of its parts, resulting in faster yet less accurate data.

A Conglomerate consists of:

An array of physically adjoining Cubes

The functions that should be enacted upon these Cube

The Core

1 The team decided to implementation optimization after attending a class at the kickoff conference in Socorro. Optimization proved to be quite useful.

The Core is the most important, albeit complex, portion of the project code.

The Core was tasked with:

Assignment, adjustment, and maintenance of Conglomerate designations

Acting upon Conglomerates with gravitational functions

Transfer data to the graphical model (Provehamus), and respond to input it may have.

Optimization is a large part of this section, and the reason why this code is structured as it is, with levels of clarity. The Core is responsible for the assignment, adjustment, and monitoring of the conglomerates, which are tools for controlling the amount of accuracy received and calculations spent. Conglomerate assignment relies on a few of the following considerations:

Disparity of mass v areas of space

Relative movement of particles

Rate of growth, in mass, of an area

User-specified input on areas of interest

The Provehamus

The Provehamus is the graphical representation of the simulation, as well as the interface to modify it. The Provehamus is where every concept comes together.

When initially creating The Provehamus, the team settled on the following criteria:

I. It must show, and be interactive around the cubical nature of the simulation. One should be able to navigate through the Cubes, and be able to clearly see their boundaries when needed. See Figure 1.

II. One must be able to interact with a Cube to find its properties: i.e., what gases and particles compose its density, its average charge, et cetera.

III. One must be able to move easily through the visualization, and not be hindered by the controls

IV. Windows should exist for all of the following: Application-wide preferences, scenario preferences, simulation control (pause/start, et cetera), cube information

V. The period of data updates should be adjustable

VI. One should be able to create scenarios graphically, with Provehamus.

The Provehamus initializes by creating a series of grid lines that act as the boundaries for the data cubes. The cubes are then rendered in 3D. These cubes are assigned components by the Core and modified by it as well. These modifications indicate how the bodies will move based on physical interactions calculated with the mathematical model. The observer can navigate the space and view the various sectors and cubes using the keyboard as well as modify the current scenario in the GUI.

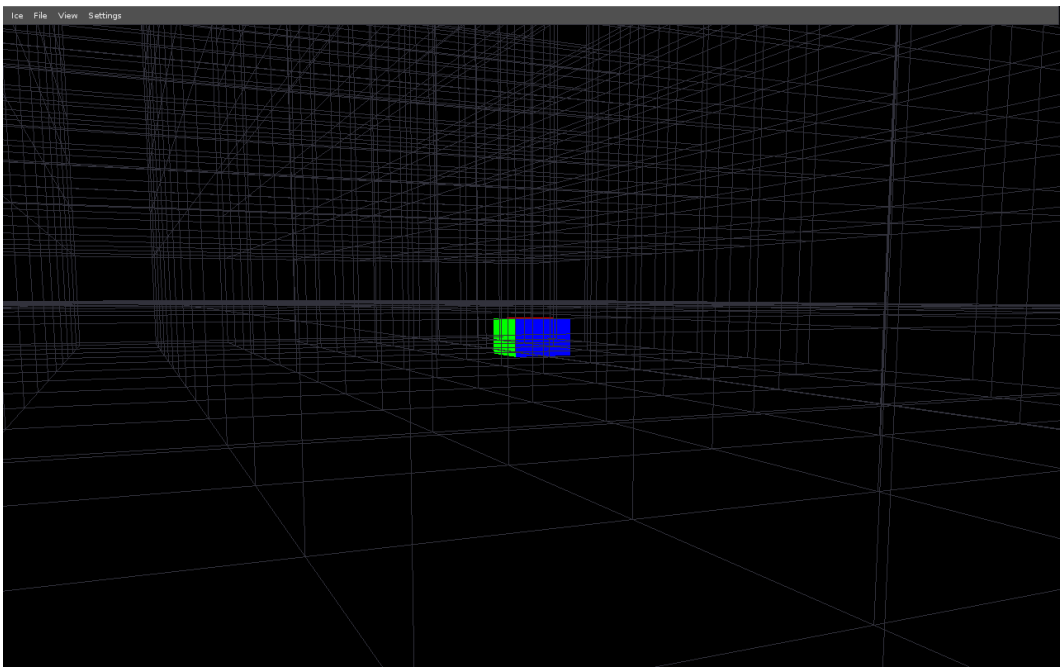


Figure 1.
The grid lines that represent the boundaries of the cubes the colorful cube is simply used as filler

The Provehamus consists of two parts: the GUI, and the OpenGL representative. The only specifics for the OpenGL part on the code, besides those listed above, are that density needs to be visually determinable, represented by a quantity of particles in a given Cube.

Original Benchmarks

The team began the project with several initial benchmarks in mind:

1. The project MUST be cross platform and Open Source at ALL points of development
2. The code must integrate multi-threading AND be optimized to run somewhat smoothly on a laptop computer
3. The project should contain real world data and the project should incorporate modularity

Meeting the marks:

Benchmark 1: Cross-Platform & Open Source

The first goal was by far the easiest to achieve. In order to keep the project Open Source, the team immediately turned to GitHub.com. GitHub allowed for the team members to keep up with each other's additions and changes to the code as well as any other documentation included on the site. Even before any code was written, documents outlining the project were uploaded. The still functional GitHub repository can be found at: https://github.com/carrollcongress/Ice

After methods of maintaining an Open Source code was implemented the team focused on making the code cross-platform. In order to make the code cross-platform, the team needed the services of a system called Cross-Platform Make (CMake). CMake allows the user of any of the major Operating Systems to

compile, link, and, in effect, build any source code locally on any machine. The team tested CMake on 4 different operating systems (Arch Linux, OS X, Windows and Ubuntu) and all operating systems compiled and rendered the code flawlessly. The CMake configuration files were then added to the GitHub repository and used by the team throughout development.

Benchmark 2: Multi-threading and Optimization

The project integrates the BOOST multi-threading library into the code. The BOOST library allows us to use multiple threads and cores based on each computer's CPU. Because the team had CPU's ranging from 2 to 4 cores, the amount of cores used had to be adaptive based on the computer. Optimization was implemented in the core section of the code. In order to optimize the code, the data cubes were summed into a large conglomerate (see Core Concept 2). A larger Conglomerate results in less calculations per area, while a smaller results in more, making this a speed v. accuracy problem. Conglomerate assignments were handled by Core.cpp.

Multithreading

This code takes advantage of multithreading concepts in order to improve the code's capabilities. There are, at a minimum, two threads running at any single time: one for the OpenGL GUI, and one for the Core. These two threads can at any time communicate via interprocess memory and message queues provided by the BOOST Muli-threading library. There are as well an optimized number of worker threads running during the simulation, to calculate

displacement vectors.

Interprocess Memory

The simulation, once loaded, is stored in a section of memory allocated by the BOOST library for inter-thread communication (See Appendix B). At this time, two threads are running in the code, the Core, and the OpenGL GUI, each independent of each other. By using BOOST message queues, the GUI can instruct the Core to begin processing the scenario data. When the Core receives this message, it locks the data, making use of an interprocess mutex, against writes from any thread.

Worker Threads

The Core, when calculating mass displacement for the gravitational model, takes advantage of the multicore architecture by spawning worker threads to process the data. A worker, in multithreaded code, is a structure that allocates its own thread, and processes data in the background for later use. Depending upon the architecture of the processor, there can be any number of "physical threads" available for use; this code maximizes performance by allocating a single worker per thread.

Benchmark 3: Real World Data and Modularity

Because the project is modeling the Interstellar Medium, the project needed to somewhat realistically represent this flow. By incorporating gravitational physics into the model, the model can give an adequate representation of the interactions

based on mass. The code needed to be modular, meaning it needs to be able to be scrutinized at the user's control. To meet this goal, the team integrated the libXML2 libraries into the code. The library allowed us to create scenarios and load them for the Provehamus and The Core to work off of. This implies user-friendliness, pushing for the need of an easily workable GUI supplied by the AGAR libraries.

The Simulation (Mathematical model)

The simulation takes into account gravitational interactions between bodies. Given that the distance *(r)* between two bodies on the Cartesian coordinate system is represented by:

$$r = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

We can then determine $F_{gravity}$, using Newton's formula for universal gravitation represented by:

$$F_{gravity} = \frac{Gm_2m_1}{r^2}$$

Where G is the Gravitational Constant = $6.67300 \times 10^{-11}\ m^3\ kg^{-1}\ s^{-2}$

given that $F = ma$ we can determine that:

$$a_n = \frac{F_{gravity}}{m_n}$$

Given that an object's position and an object's acceleration are related by:

$$position_{final} = \int (at + v_i)dt$$

We find that

$$\int (at + v_0)dt = \frac{1}{2}at^2 + v_0 t + position_{initial}$$

Therefore

$$\Delta position = \frac{1}{2}at^2 + v_0 t$$

After finding how our force vectors relate, we can determine how the position of the affected object changes as it interacts with the other bodies in the simulation. With this we can change the objects x,y and z position accordingly.

The implementation of Newton's Law of Universal Gravitation and modification of the x,y,z coordinates are found in Core.cpp:

```
dist =
pow(time,2)*(gconst*(*it).second.mass())/(2*pow(p,2));
```

In order to facilitate interactions between bodies in the simulation, it was necessary to convert from the Cartesian coordinate system to the spherical coordinate system. Conversion goes as follows:

$$\rho = \sqrt{x^2 + y^2 + z^2}$$

$$x \to \rho sin(\phi)cos(\theta)$$
$$y \to \rho sin(\phi)sin(\theta)$$
$$z \to \rho cos(\phi)$$

with the coordinates being related such that:

$$tan\theta = \frac{y}{x} \qquad tan\phi = \frac{\sqrt{x^2 + y^2}}{z^2}$$

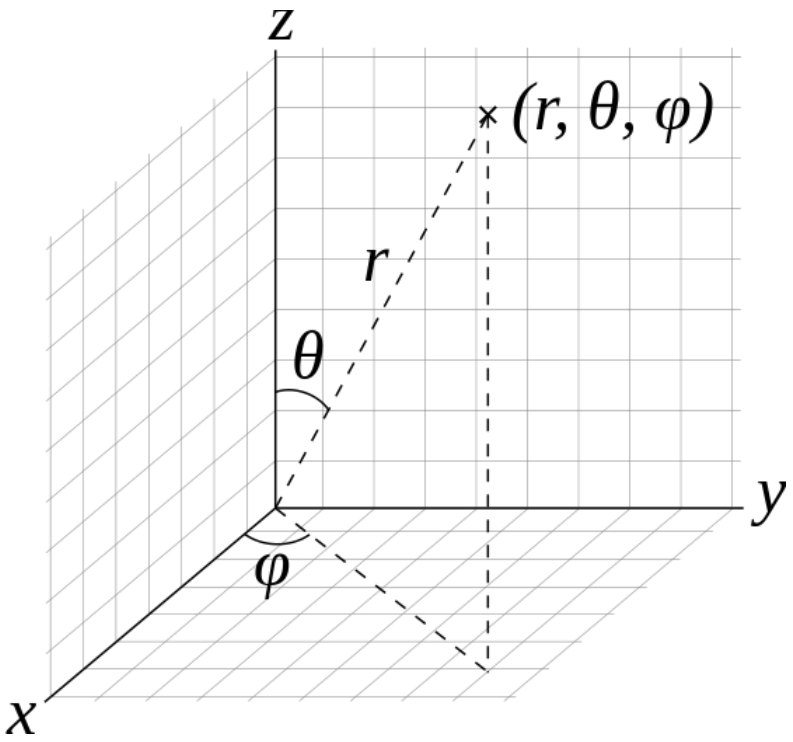


Figure 2: a graphical representation of the spherical coordinate system (source:en.wikipedia.org)

The spherical coordinate system allowed the simulation to break down each force vector into its three dimensional components and adjust position in space accordingly. Application of the spherical coordinate system can be found in Core.cpp:

```
rho = pow(pow((*it).second.location.x-data-
>location.x,2)+pow((*it).second.location.z-data-
>location.z,2)+pow((*it).second.location.y-data-
>location.y,2),.5);
theta = atan((((*it).second.location.y-data-
>location.y)/pow(pow((*it).second.location.z-data-
>location.z,2) + pow((*it).second.location.x-data-
>location.x,2), .5));
phi = atan(((*it).second.location.z-data-
>location.z)/((*it).second.location.x-data->location.x));
```

5. Results and Conclusion

The computations conducted in the Core allowed us to achieve results that would not have been attainable in the standard N-Body simulation. The team's code is able to conduct its computations more efficiently through optimization through approximation of various sectors of space. This aspect is important as it allows us to simulate the movement of diffuse areas of matter.

Because our code does not incorporate charge and thermodynamics, it is not yet accurate to claim that it is adequate for conduction of tests concerning formations of nebulae. However, our model has proven to be much more efficient and accurate for *diffuse* areas of large amounts of matter when compared to other N-Body simulations. Because of this, the code has the potential to become increasingly accurate as improvements in the physical interactions are made. This does not mean that the current code is incomplete, it simply only incorporates mass and density but can easily be adaptable to incorporate charge and thermodynamics.

6. Future Work

Any future work on our model would likely be to improve its modularity or adaptability. Because of our open-development process, anyone can view the code on GitHub and can replicate or modify our model easily. More scenarios for our model to load could be added, allowing a user to adapt it to a number of real-world interstellar medium formations.

7. References

Knight, J. (n.d.). Nebulae - Celestial Objects on Sea and Sky. *Sea and Sky - Explore the Oceans Below and the Skies Above*. Retrieved April 1, 2012, from http://www.seasky.org/celestial-objects/nebulae.html#Section%202

Planetary nebulae - Scholarpedia. (n.d.). *Scholarpedia*. Retrieved April 2, 2012, from http://www.scholarpedia.org/article/Planetary_nebulae

University of New Hampshire Experimental Space Plasma Group. (n.d.). What is the interstellar medium?. *UNH Experimental Space Plasma Group*. Retrieved April 1, 2012, from http://www-ssg.sr.unh.edu/ism/what1.html

9. Acknowledgements

Michael Harris

Ben Mitchell

John Klosterman

Terrance LeBeck

Fredrick Bobberts

Appendix A: CMake Configuration

As mentioned previously, the project was designed around being cross platform. Because of this, we needed CMAKE to ensure flawless transfer and compilation over each operating system. The following is the configuration file CMakeLists.txt:

```
cmake_minimum_required (VERSION 2.6)
# The version number.
project (Ice)

set (Ice_VERSION_MAJOR 1)
set (Ice_VERSION_MINOR 0)

set(CMAKE_ALLOW_LOOSE_LOOP_CONSTRUCTS true)
set(CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR})

# configure a header file to pass some of the CMake settings
# to the source code
configure_file (
  "${PROJECT_SOURCE_DIR}/IceConfig.h.in"
  "${PROJECT_BINARY_DIR}/src/IceConfig.h"
  )

Find_Package ( SDL REQUIRED )
if ( NOT SDL_FOUND )
message ( FATAL_ERROR "SDL not found!" )
endif ( NOT SDL_FOUND )

find_package(OpenGL REQUIRED)
find_package(Boost COMPONENTS system thread filesystem REQUIRED)

execute_process(COMMAND agar-config --cflags OUTPUT_VARIABLE AGAR_CFLAGS)
execute_process(COMMAND agar-config --libs OUTPUT_VARIABLE AGAR_LFLAGS)

add_definitions(${AGAR_CFLAGS})
set(CMAKE_EXE_LINKER_FLAGS "-L/usr/lib -lag_gui -lag_core -L/usr/lib -lfreetype
-lz -lbz2  -L/usr/lib -lm -L/usr/lib -ljpeg -L/usr/lib -lpng15 -lpthread -lrt")

if(APPLE)
set(ALL_OSX_SRCS src/osx/SDLMain.m src/osx/SDLMain.h)
set (OSX_HEADERS "/opt/local/include")
set(CMAKE_EXE_LINKER_FLAGS "-L/usr/local/lib -lag_gui -lag_core -
L/opt/local/lib -Wl,-framework,Cocoa -L/opt/local/lib -lfreetype -lz -lbz2 -
framework OpenGL -lm -L/opt/local/lib -ljpeg -L/opt/local/lib -lpng14 -
lpthread")

include_directories(
${OSX_HEADERS}
)
endif()

# add the binary tree to the search path for include files
# so that we will find TutorialConfig.h
include_directories("${PROJECT_BINARY_DIR}" ${SDL_INCLUDE_DIRS})
FILE(GLOB ALL_CPP_SRCS src/*.cpp)
FILE(GLOB ALL_H_SRCS src/*.hpp)

add_executable(Ice ${ALL_CPP_SRCS} ${ALL_H_SRCS} ${ALL_D_SRCS} ${ALL_OSX_SRCS})
```

```
target_link_libraries (Ice
${SDL_LIBRARY}
${GLUT_LIBRARY}
${OPENGL_LIBRARY}
${Boost_FILESYSTEM_LIBRARY}
${Boost_SYSTEM_LIBRARY}
${Boost_THREAD_LIBRARY}
)
```

Team Note: Ubuntu Linux continued to have trouble linking against the AGAR libraries when using CMake. CMake worked fine on other operating systems. As a workaround, Louis Jencka developed a shell script ubuntu.sh. The script simply uses gcc to manually compile and link the code:

```
cd src
gcc -o Ice -I/usr/include/libxml2 -I/usr/local/include/agar -I/usr/include/SDL
-D_GNU_SOURCE=1 -D_REENTRANT  -I/usr/include/freetype2   -
I/usr/include/libpng12 main.cpp Provehamus.cpp Core.cpp -L/usr/lib/ -lSDLmain -
lSDL -lpthread -L/usr/lib/x86_64-linux-gnu/ -lGLU -lGL -lSM -lICE -lX11 -lXext
-L/usr/lib/ -lboost_filesystem-mt -lboost_system-mt -lboost_thread-mt -
L/usr/lib64 -L/usr/local/lib -lag_gui -lag_core -L/usr/lib -lSDL -
L/usr/lib/x86_64-linux-gnu -lfreetype -lz  -L/usr/local/lib -lGL -lm -L/usr/lib
-ljpeg -L/usr/lib/x86_64-linux-gnu -lpng12 -lxml2
```

Appendix B: An Example Scenario File

The following is a general layout of our scenario file (.ice extension). These scenarios are, at the core, simply XML Scenarios:

File.ice:

```
<ice name="A" X="x(cubes)" Y="y" Z="z" length="U(meters)">

<mass name="A" mass="x" charge="y" id="1"/>
<mass name="B" mass="x" charge="y" id="2"/>
<mass name="C" mass="x" charge="y" id="3"/>

<cube X="x" Y="y" Z="z">
      <medium mass="x" id="2"/>
      <medium mass="x" id="1"/>
</cube>
<cube X="x" Y="y" Z="z">
      <medium mass="x" id="3"/>
      <medium mass="x" id="1"/>
</cube>
<conglomerate X="x" Y="y" Z="z" XL="x" YL="y" ZL="z">
      <medium mass="x" id="3"/>
      <medium mass="x" id="1"/>
</conglomerate>
</ice>
```

Appendix C: Scenario Data Structure

```
class dataplexor {
    public:
    //Scenario Name
    std::string simulationName;

    //A side's length in meters
    double cubeSize;

    //A side's length in Cubes
    vector size;

    //The array of cubes used.
    std::map<vector,cube> cubeArray;

    //The array of conglomerates used.
    std::vector<conglomerate> optArray;

    //The array of mass types available
    std::map<int,dust> dustArray;

    //Multithreading Mutex
    boost::interprocess::interprocess_upgradable_mutex
mutex;

    //Cube access functions
    cube* cubeAt(vector loc);
    cube* cubeAt(int x, int y, int z);

    bool halt;
};@
```