GLNebula Manual

Juan José Aja Fernández - juan.aja@gmail.com

4 de julio de 2006

1. Introduction

GLNebula is a system designed for the real-time visualization of planetary nebulae models (and in general, any static one-component scalar field). produced by three-dimensional photoionization codes such as: Cloudy_3D[1]. It uses three files: each one represents the emission values of one ionized element inside the nebula. The files are arranged in a **cubical grid** with one scalar per cell, representing the emission value. Each file is matched to a color channel so that the scalar values represent the intensity of a particular light color, also the cells in the final model have color blending capacities as well as transparency to provide the plasma-like look characteristic of their astronomical counterparts. GLNebula uses 2D camera-oriented particles to construct the model of the nebula and give it the illusion of being volumetric, thanks to this fact the performance is very acceptable (as opposed to use proper cubes to represent each cell).

Currently it uses two types of particles: camera-oriented quads (known as billboards) and point sprites. The latter requires a fairly modern video card, specifically one that supports the extension: GL_ARB_point_sprite[2], and is considerably faster than the billboard approach[3]. Unfortunately one vendor of video cards (ATI) hasn't complied fully with the OpenGL standard, so the extension may or may not be present, and is usually broken (for instance in the Radeon 9800 models, plain point sprites can be displayed, but texturing them isn't supported, so if you have an ATI card and the model appears dark or doesn't appears at all you could try disabling textures).

For an in-depth explanation of how GLNeblua works (and if you speak spanish) please let me know and I will send you a copy of my thesis.

Also if you are interested in the real-time visualization of dynamic scalar fields (such as flow visualization), a version which handles time and change within the fields is currently on the works.

[1] $http://132,248,1,102/Cloudy_3D/$

[2] This can be checked with the glewinfo program, bundled with the glew library required to run GLNebula and in the windows version zip

[3] Moreover the grid using point sprites can be encapsulated in a display list, so the permformance is even better

2. Features

- **Real-time visualization of medium to large realistic models**: Due to the use of bidimensional particles to create the effect of a 3D volume, GLNebula offers enough framerates as to provide interactivity with the user (meaning no large time gaps between user events). It has been tested with models up to 250x250x250 (albeit with 500 thousand visible cells) with satisfactory results. Also by means of texture mapping, color blending and particle transparency the resulting models can look fairly similar (at least to my perception, I'm not an astronomer) to the real ones.
- **Free camera control**: A camera has been implemented in GLNebula to allow the user free exploration of the model, basic camera movements (zoom in/out, model rotation) are done with the mouse.
- **Parameter control**: GLNebula offers full control of the visualization parameters to the user, so that: transparency, type and size of the particles, color scaling and scalar threshold value can be adjusted and the changes made apply in real time (in the case of billboards, with sprites a button must be pressed).
- **Session management**: In order to speed up the process of reconstructing previous visualizations, GLNebula has a simple session managment system which can save the current state of the model or load an existing one. Also it can take screen captures of the main window to png images with the possiblity of writing a caption to the final image containing the name of the session, the dimensions of the grid and the datafiles representing the colors.

3. File structures

The data files must contain only ASCII characters (numbers) and no other data but the scalars must be present. They **MUST** be **CUBES**, not cubical grids are not supported for now, also they must have the same dimensions between each other.

GLNebula uses a particular file structure for its sessions:

- Grid dimensions
- Red layer filename
- Green layer filename
- Blue layer filename
- Red scaling (red saturation)
- Green scaling (green saturation)
- Blue scaling (blue saturation)
- Scalar value threshold

- Transparency (alpha channel)
- Particle size: 0 = camera-oriented quads, 1 = point sprites
- Texture filename
- Enable/Disable center mark
- Enable/Disable axis
- Enable/Disable texture mapping
- Enable/Disable particle blend

Be wary that the order matters, so expect unusual behavior if you manually edit the file without care.

The "standard" extension of the files is .neb, but it isn't a must (for convenience the filter in the load session file chooser window is set to *.neb, but can be changed to display a session file with other extension).

4. Requirements and installation

The easiest way to run GLNebula is under windows, by downloading the zip containing the executable and the needed .dlls and by double clicking on the application.

The source code compilation requires several libraries and their respective development headers to work, in no particular order:

- **OpenGLL**[4]**, GLUT**[5]
- **GLEW**[6]
- **Fast Light Toolkit (fltk) 1.1.7**[7]
- **Simple Direct Media Layer (SDL)**[8]**, SDL_Image**[9]
- **GD Graphics Library**[10]

Also a modern video card is greatly recommended, the performance increases a lot and today's commodity hardware prices are relatively low (in future versions GLNebula will use shaders, which will allow the visualization of very large models with much more detail and performance and are only present in modern GPUs).

[4] *http://www.opengl.org/*
[5] *http://www.opengl.org/resources/libraries/glut.html*
[6] *http://glew.sourceforge.net/*
[7] *http://www.fltk.org*
[8] *http://www.libsdl.org/index.php*
[9] *http://www.libsdl.org/projects/SDL_image/*
[10] *http://www.boutell.com/gd/*

4.1. Linux compilation

Compiling it's as easy as: *./configure*, *make* (the binary will be in the src/ directory) and optionally *make install* to copy the binary into /usr/local/bin. The Makefile bundled with the tarball will try to find the development headers in their respective directories under /usr/local (for instance the OpenGL header must be in /usr/local/GL/gl.h, and so on), so edit it accordingly if that is not the case.

Also there's a .kdevelop file, just open it under kdevelop and build the project, it should go without trouble).

Look for distro-specific binaries (Fedora, Suse, Ubuntu, Debian) in the near future, also let me know of any trouble with the process.

4.2. Windows compilation

For windows there is a DevC++[11] project file, just install the appropriate devpaks (all available through the community devpaks in the package manager) and build the application. (sorry MS Visual Studio users: no .dsps for now). As soon as I have access to MacOS X I'll make a binary, for now I believe that the Makefile it's fairly similar, so I hope it works with minimal modifications.

5. Using GLNebula

5.1. The control window

Here's where the visualization parameters go, let's review each control:

- **Menu**: The menu has four sub-items:

  Load Session: Displays a dialog to select the session file to open

  Save Session: Saves the current session

  Save Session As: Displays a dialog to select the filename to save the current session in

  Quit: Exits the program

- **Load Buttons**: When pressed, the buttons at the leftmost part of the window will open a dialog that will let you choose a file to load: the first one will load a session (similar to the Load Session menu item), the next one let's you choose the texture (bitmap) file to apply to the particles (it's entirely optional, GLNebula can work without it), lastly the remaining three correspond to the data files (in RGB order). The path to the files selected will appear in the boxes to the right of each button. When all three data files have been loaded, push the Update Data/Cut/Textures button to display the model in the main window (this step isn't necessary if you loaded a session: the moment.you select the file and press OK the data will be loaded automatically).

[11] http://www.bloodshed.net/devcpp.html

- **Text boxes**: The first five boxes only display the path to the files selected with their respective load button. The last one, labeled *Cut Value*, is editable. In this box goes the minimum value that the scalars in the data must have, so for example if the cut value is set to 1e-19, the cells with scalar value less or equal to 1e-19 will be considered invisible and will not be displayed. Setting this box to a proper value can increase performance considerably, note that when the data is loaded the console (presented a few items below) will display the maximum, minimum and average values to select possible candidates for threshold values.

- **Parameters**: These sliders control the diferent parameters that define a visualization.

  Alpha: This slider will control the transparency, going from 0 to 1 stepping by 0.01 every time the little arrows are pressed.

  Particle Size: This widget modifies the particle size in the model, the maximum size is tied to a parameter in every video card so beyond certain point (usually 60 but i've seen bigger point size) is impossible to increase their size.

  Scaling: The next three sliders control the color scaling (saturation) of the visualization, so if for instance you would like to increase the presence of a particular ionized element, just increase the value of the color corresponding to the proper data layer, similarly if you want a layer to dissapear simply set the appropriate scaling factor to zero.

  Sampling: This slider controls the data sample rate in powers of 2, so if it's set to 1 only half of the cells in the grid will be displayed, if set to 2 only 1 in every 4 cells will be displayed, and so on.

- **Misc. checkboxes**: These control the displaying of the center mark and axes, also enable/disable texture mapping and blending.

- **Particle type radio buttons**: Mutually exclusive, they determine the type of particle to be displayed.

- **Update buttons**

  Update Data/Cut/Textures: This button reconstructs the grid, updates the threshold value and loads the textures.

  Update Params/Scaling: When using billboards the changes made in the parameters take effect instantly, so this button is greyed out. But when using point sprites the parameters change won't reflect in the visualization until this button is pressed.

- **Snapshots**: The snapshot button will save the current session and then will write a png image of the current visualization. It's important to first save the session or to have a session name defined, otherwise the image won't be written. The checkbox above the button will enable/disable the writing of a caption containing the grid dimensions, data files and session name to the image.

- **Console**: Here's where all the program output is printed, every error and message is displayed here. Errors are preceded with asterisks for easy detection.

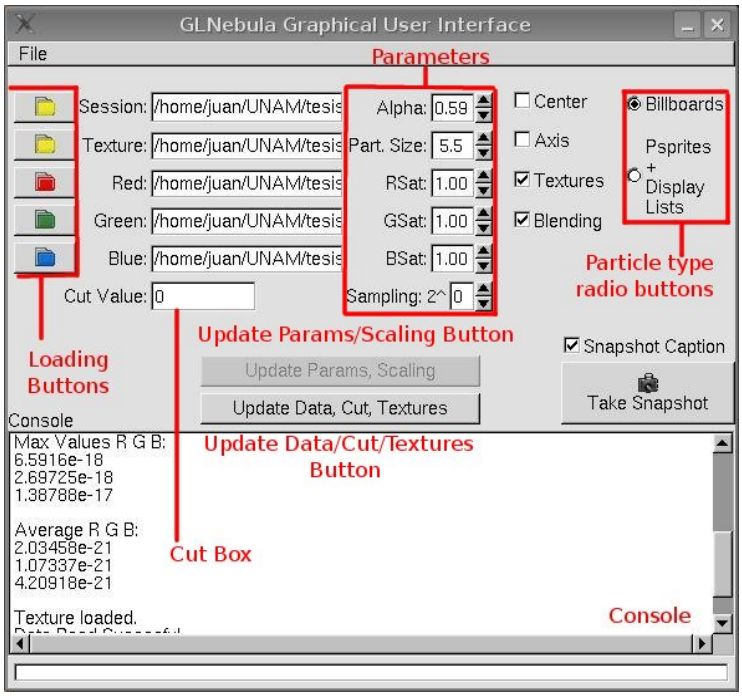

Figura 1: GLNebula control window)

.

5.2. The main window

Here's where the model will appear when the data is loaded and where the camera controls are. To rotate the model simply click the mouse left button anywhere inside the window and while still pressing it, drag the pointer towards the direction of the desired rotation. To zoom in or out just press the right mouse button and drag it (upwards to zoom in, downwards to zoom out). Zooming can also be performed with the mouse wheel.

6. Future features

Hopefully with more wide spread use many feature requests will appear, for now these are the future features that GLNebula will have:

- **Further optimizations**: To speed up the rendering GLNebula will use Vertex Arrays and Vertex Buffer Objects, features that require a more powerful GPU but will provide greater frame rates (and consequently larger models could be used).
- **Filters**: Currently I'm implementing a simple sqrt filter to normalize the transparency in the model, but unfortunately is very slow. In the future I plan to use shaders (this requiring a more powerful GPU, so it will be optional) to make more filters and preserve real-time framerates.

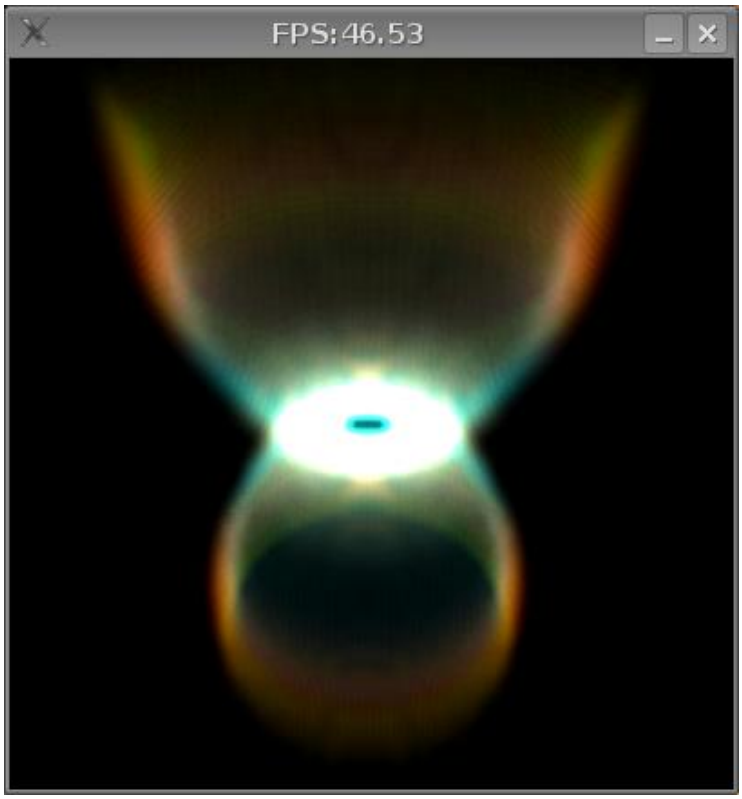

Figura 2: GLNebula main window displaying a model of MyCn18 (Data courtesy of Christophe Morisset, IA UNAM)

- **Visualization of dynamic scalar fields**: By introducing the notion of time it will be possible to load n files (probably by specifiyng a common header or preffix) representing n states of a scalar filed across a period of time, and to provide control over the animation by a slider bar which will control the timeflow, all this in real-time. Hopefully this feature will attract the interest not only of astronomers but anyone interested in flow visualization.

7. Bug reports, feature requests, questions, complaints, etc.

Contact me: juan.aja@gmail.com, I'll respond as soon as possible.