Adam Susser

**Nbody Blender Visualizer**

**Introduction**

When performing science, it is generally useful to be able to see what is going on physically. This is especially the case for science done using computer programs, which can be thrown way off by programming bugs. The nbody blender visualizer is designed to alleviate this problem, and to allow scientists to better see what their computers’ number crunchings mean.

Currently the nbody blender visualizer is designed to work with the milkyway\_nbody program, though it can theoretically be changed to work with other programs. Milkyway\_nbody is a program created to simulate tidal disruptions of dwarf galaxies, and then to compare the results to observed data. To accomplish this it creates “particles” which represent collections of stars in a dwarf galaxy. The dwarf galaxy is then made subject to a gravitational model which simulates the Milky Way’s gravitational potential. Differential forces cause particles which are closer to the galaxy to be pulled harder by the galaxy, and thus these particles move more than more distant particles. This causes the dwarf galaxy to be pulled into a long, stream-like shape, hence the name “tidal stream”. The final stream is then compared statistically to observed streams. This is done many times with differing parameters, such as light-dark matter composition, which control the pre-disruption dwarf galaxy. The objective of the project is then to see if the program can determine what the “right” parameters were which lead to observed tidal streams, and thus to learn such things as the light-dark matter compositions of real objects.

**Using the blender visualizer**

The nbody blender program is relatively easy to use. It should be even easier to use via the record.sh shell script, but unfortunately blender 2.66 segfaults when used in a certain necessary way in a script. Thus some manual work must be done, but it luckily isn’t too much. First, one should make sure they are using the version of nbody that supports blender. This version can be found at <https://github.com/sussea/nbody_blender_visualizer> . The three important files here are the nbody\_plain.c, nbody\_cl.c, and blender\_visualizer.c. Assuming nothing radical has changed since these were written, it should be possible to throw them into a more recent version of nbody. If such radical changes have been made, there is no need to worry – the version of nbody on the above linked-to git will work.

Once nbody is ready, one only needs to run the milkyway\_nbody program normally. Output in the form of “blender\_events\_record.txt”, “blender\_orbit\_record.txt”, and “blender\_misc\_record.txt” will be produced in the current directory. These files store all the information that the blender visualizer will need. It is a decently good idea to have some idea what these files contain. The events record contains all locations of all particles across all of the time of the simulation. The orbit record contains the full path of the orbit, generated by concentrating all the particles in one central location. The orbit path becomes the orbit tracer in the visualizer, a colored curve which grows longer as the simulation progresses. The misc record stores miscellaneous information, such as the total time of the simulation, how many particles there are, and how many frames to animate. It also stores six coordinates which are used to determine the approximate plane of the tidal stream’s orbit, which is useful for getting the camera to point in the right direction to see the entire disruption from a perpendicular angle. The final portion of the misc record stores information on which particles are dark matter and which are light matter.

These files should be in your current directory when you call blender so that the program can find them. There should also be an empty directory called “pics” in your current directory. The blender program can be opened using the command “blender some\_directory/nbody15.blend”, where some\_directory is the location of the nbody15.blend file. If the file is in your current directory, you can simply use “blender nbody15.blend”. The blender editor window should open up. The majority of the information it shows is unnecessary for someone who just wants to use the nbody15.blend program as it is, though it may be fun to play around with. The important screen in the blender window is the one on the far left with text on a gray background. This screen holds most of the actual program. At the bottom, there should be a little text box that says “Main”. If it says anything else, you are looking at one of the bonus programs which editors can use to easily manipulate stuff (such as by removing all particles). To the right of the “Main” text box there should be a button which says “Run Script”. Click it, and wait. The program may need to run for a few hours. You are basically home free once the program finishes, in that there is one final step to your recording.

Blender is currently rigged to create not a movie, but a bunch of png files. This is in part due to suggestions found on a blender tutorial. It is possible to change it to record a movie directly, but it is safer from corruption and easier to manipulate if this blender option is kept as is. Instead, a third program should be used which converts pngs to movies. In the record.sh shell script there is a command which uses ffmpeg. Edit this command to fit your directory scheme, though it should be fine as-is if you are in the directory directly above “pics”. This will turn the pngs into a movie in a matter of seconds. The default movie name is out.mov, though you can change this to anything you want. By default h264, a high quality video codec, is used. You may change it to something else depending on your needs.

**Known bugs**

The 3D galaxy model may freak out and create weird orb things on a frame or two very early in a visualization (before frame 10). At the moment there is no known good fix for this, just remove the offending frames from the pics directory before running ffmpeg.

If you are using the flat Milky Way model (not used by default), you may run into an annoying glitch. This glitch will turn the representation of the Milky Way into a pink square. This is due to the program not being able to find the Milky\_Way.png file. This can be fixed by using the “export image to plane” addon which comes with blender. The addon will annoyingly need to be activated before it can be used. To activate it, select file->user preferences. Choose the addons tab, and type “image” into the search bar at the far left under the tabs. It should be one of the options. Check the box and the addon should activate. Now go into file->import->Images as Planes. Scroll all the way down with the far-left scroll bar, and find the box labeled “Material Settings: (Blender)”. Select “Use Alpha”. Next, look to the box labeled “Plane dimensions:”. Select “Absolute”. Keep all other settings the same. Find and select the image of the galaxy you want to use, and import it. To make it the right size, you will want to change its scale. It should already be selected as a result of being imported, but if it is not then select it with right click. On the far right of the screen should be the properties editor (the outliner editor should be right above it, and perhaps look like part of it). Select the orange cube-looking tab, and information on the object’s location and scale should appear. Change the first two options in scale to 32, and change all values under location to 0. The value of 32 is used instead of 30 because the image is smaller than the plane we are scaling. Choose the checkerboard-like tab next. There should be three sub-tabs near the top now – a red sphere with four parts, an Earth-looking tab, and a paintbrush tab. Choose the red sphere. Scroll down until you see “Image Mapping”. Click this to open the image mapping sub-sub-tab. Now click the extensions drop down list and choose “Clip”. Next, click the “Influence” sub-sub-tab. Under the column on the right there should be an option labeled “Emit”. Click it, and enter 1. Now choose the red sphere at the top where the orange cube was to enter the “Material” tab. Scroll down to “Transparency”, and under that choose “Z Transparency”. If the alpha in this section is set to anything but 0, set it to 0. Now delete the other Milky Way. Finally, go into the outliner editor and right click the new Milky Way (which should be named something like Milky\_Way.jpg001) and rename it to “Milkyway”. The bug should now be fixed.

**Modifying the program**

There may be need to edit the blender program. On the nbody end, the program merely spits data into the record text files. However, the order it does this in is very important. The camera’s 6 coordinates are taken from two center of mass locations. One is the initial center of mass of the tidal stream, and the other is a center of mass approximately 90 degrees away along the plane of the center of mass path. Using these two vectors, it is possible to create an approximate tidal plane. When printing particle color, the program uses the “ignore” function. If a particle should be ignored in some data, then the ignore function returns a 1. Dark particles are the ones which are sometimes ignored. Thus a 1 represents a dark particle, and a 0 represents a light particle.

The blender end of the setup may be a little trickier to work with. Though much of the data an editor would need to see is in the text screen, *not all of it is*. There are several non-particles in the program, each which are pre-initialized with data independently from the script. These include the camera and its HUD elements Progress and XYZ\_legend, the 3D galaxy model and the fields which control it, the 2D Milk Way model, an orbit tracer, some yellow and purple particle templates, and some axis elements, and an unfortunate hack called progress offset. The camera will come pre-generated with the proper vision range. If anything becomes further away than the vision range, it will need to be increased. The vision range is labeled “clipping”, and is located in the properties editor. The orbit tracer starts out as an empty mesh with basically no information worth knowing. The 3D galaxy model is a particle emitter whose emitted particles are shaped by the vortex fields Field1 and Field2. The 2D galaxy model is very basic, aside from what was discussed early with respect to the 2D galaxy bug (though it's not really a bug so much as an inconvenient download issue). The HUD text has two important types of information stored accessible in the properties editor. One is the font/text size. Just click the F tab in the properties editor to change that, though some of the script may need to be changed to keep the text from leaking out of the camera’s view. There is also a constraint put on the HUD text objects. They are the only objects with any constraints. The constraint forces them to always be located a local position transformation away from where you’d otherwise expect it to be. This is a hack way to cause local position transformations, and probably not the best way. Because one cannot create a constraint with pure numbers, these numbers have to be attached to an object. Thus the progress offset and XYZ legend offset objects, which are empty invisible unrenderable object whose sole purpose is to have a position which HUD objects can see. The yellow and purple templates serve a similar purpose – they exist only to be copied in the initialize\_particles() function, and never do any work in the foreground like would be expected from an object.

With all that out of the way, we can focus on the script. The only script which you need to use is the Main script – the rest are either helpers for debugging or places where I toyed with new features. At the top are mode options for how the video will be generated. Below those are definitions of functions. Under those is the main program which calls these functions, and below that is a somewhat separate program used to revolve the camera for a fuller 3d view of the final stream. For convenience, references are set up which can quickly access such objects as the camera. Data is read and parsed from the events, orbit, and setup text file accessors. First setup is read through. The function initialize\_camera() uses the 6 coordinates and some geometry to create a vector perpendicular to the plane of the tidal stream. The camera is moved a reasonable distance along the perpendicular vector, and made to point along the vector. The HUD text is then moved right in front of the camera, which will make it appear in the bottom corners thanks to the local offset constraint hack. The particles must then be initialized. If the circle template is used, then they must be facing the camera before the camera takes pictures of them, or else they may be perpendicular to the camera and thus invisible due to their complete flatness. Thus the templates are rotated so copies are initialized with the correct orientation. They are then copied from either the dark or light matter template, depending on the type. They are given sequential names because blender orders bpy.data.objects alphabetically because it is a jerk. Because of the way blender determines alphabetical order, 20 will come right after 2 if we're not careful. The current fix is adding a big enough power of ten to ensure they all have the same number of digits. In addition, the orbit tracer must be updated. This is done easily with the add\_to\_orbit\_tracer() function which opens up the object editor, adds a point to the orbit tracer curve, and then closes the object editor. To ensure nothing else is also edited, the function must deselect everything beforehand. When all initialization is done, the bulk of the program’s work can begin. The bulk of the work is done by a for loop which steps backwards through all objects. It is set to step backwards instead of forwards because blender automatically messes with the order of objects in bpy.data.objects because it thinks doing so is funny. This loop updates positions, takes pictures, and then updates positions again. Pictures are taken with the snapshot() function, which is another hack. This hack allows for automatically ordered single-image png slideshows. There is probably a better way to do this. Once this loop is finished, the camera revolving sub-program can initiate. This segment of the script will revolve the camera using geometric magic, rotating the camera back toward the Milky Way at each step. Because the optional starry background whizzes by “like you’re in a millennium falcon” as one group member said, it has been made to automatically turn off when recording the revolution.

**Bypassing png creation**

If you wish to bypass making png’s and go straight for movie generation, this can be done by going into the properties editor and choosing the first sub-tab (which looks like a camera). Near the bottom should be an output sub-sub-tab. Click it, and change the output type from png to whatever you want.