TAMING THE BOX TREE SYSTEM

Randy Hall, Baylor University Department of Physics, July, 2009

The Box Tree System is a collection of computer programs written in C in the early 1990's by Derek C. Richardson as part of his PhD thesis at the University of Cambridge. The system of programs is copyrighted by the author but is distributed under the terms of the GNU General Purpose License so that it may be freely used and modified. The software is used as a simulator for planetesimal dynamics and is described in detail in the thesis paper "Planetisimal Dynamics" which is available via the internet. The version used by CASPER at Baylor University has been enhanced by various people over a period of at least thirteen years and is used as a part of research projects by a variety of students and staff at the University. The software is generally well written and functions to a high level of correctness, but there are certain characteristics of the code that make it difficult to read and modify and, over the years, the code has apparently become infested with a number of bugs and or question marks regarding functionality. These issues make continued enhancement of the system difficult, as it is a challenge to decipher the existing code and to work around sections of the code that may not work properly.

A process has been started to modify the Box Tree code to make it more readable and less difficult to modify. To date, three phases of this modification have been completed. First, function, external data, and program cluster names have been modified to include a numbering scheme to aid in location of function and data definitions. Second, data structure names have been modified to include a prefix that clearly identifies a member data item as belonging to a particular data structure and the data structure definitions have been moved to individual files. Third, a GUI front end has been added to the system to allow for a user friendly method for system parameter declarations.

Function, program, and data name modifications

Box Tree contains some 30 program files, most of which contain multiple functions which (by being declared as external) may be accessed from any of the other program files. In addition, some program files contain global (external) data declarations. While there is a certain logic with regard to the placement of these items into the program files, it is not easy to find the code for a function or the declaration for a data item when it is encountered as a called function or referenced data item somewhere else in the code. To facilitate location of these external functions, each program file has been given a five character prefix. The first two characters of this prefix are bt(for Box Tree) and the final three characters comprise a number which uniquely identifies that program file. Within the program file each external function or data item has had this prefix attached to its name. So, for example, function "PointMass" within the program file "potential.c" has become function "bt610_PointMass" within the program file "bt610_potential.c." This simple change allows a programmer who encounters this function to realize immediately that it is an external function and that the code for it lives in the "bt610_potential.c" file. Furthermore, any function or data name encountered which does not include one of these prefixes can be assumed to be local.

Data structure name modifications

Box Tree makes use of several data structures which are declared to store related data together. These structures are global in nature and are defined in the main header file for Box Tree, but when the individual data items are encountered in a program it is often difficult to discern the structure to which the data item belongs or to find the data structure definition itself. This problem is compounded by the fact that the same data name can be used in multiple data structures, which can lead to confusion when encountering one of these data names with multiple definitions (This confusion is resolved by determining to which data structure the data item belongs). To alleviate this issue, the names of all global data structures have been modified to include a four character prefix. This prefix contains a one character type field ("s" for struct, "u" for union) followed by a three character identifier unique to that data structure. This same prefix has also been added to each data name contained within the structure. So, when a programmer encounters the data name "vel" in a program, there is no longer confusion over whether this data item is in the "DATA_T" struct or in the "NODE_T" struct, because the data item will now appear as either "sdat_vel" or "snde_vel," depending on whether the data item lives within the "SDAT_DATA_T" struct or the "SNDE_NODE_T" struct.

To further the ease of finding one of these data items, these data structure definitions themselves have been pulled out of the main header file and placed in individual header files which are then included individually in the main header file. This allows a programmer searching for a data structure definition to go directly to the header file for that data structure rather than having to search through the main header file. The naming convention for these header files is a "bts" prefix (for Box Tree System) the type indicator ("s" or "u") and the three character data structure identifier. So, for example the "snde" data items that live in the "SNDE_NODE_T" data structure have been placed in the "btssnde.h" header file.

It should be noted that all these changes are name changes only and will not affect execution time of the system in any way.

User friendly runtime parameter setup

Box Tree is driven by a collection of user specified parameters which are read into the Box Tree System from a sequential text file. This parameter file contains some 150 different parameters and, as such, can be overwhelming to create correctly. The parameter file is edited after being read and errors in the parameters can cause Box Tree to abort its run or to run (sometimes a very long time) incorrectly. To help facilitate the setup of this parameter file, a front end program has been developed to allow user entry of these parameters in a more controlled environment than via a text editor. This new program has been developed in Java, an object oriented interpreted language in wide use for web based and other non-performance critical applications. The interpretive nature of Java means that it is not well suited for compute intensive applications but is ideal for data entry type applications such as this. The user interface is GUI in nature and a software add on called BreezySwing has been used to implement the GUI controls. BreezySwing is a freeware product written by Martin Osborne of Western Washington University and Ken Lambert of Washington and Lee University that, with some restrictions, allows for a much simpler set up and use of GUI controls than having to deal directly with the Windows API through the standard Java classes.

The design of the front end program allows for complete editing of each parameter if desired, but the idea is to provide a collection of basic run scenarios (for which default parameter values have been established) and then allow the user to modify only those handful of parameters that might vary from run to run. In so doing, the parameter setup issue is reduced to identification of a base scenario and then individual editing of a small number of parameters.

Another advantage to this approach is that it allows online editing of most parameter values to catch errors before the Box Tree System execution begins. For parameters with a small number of valid values this is accomplished through the use of checkboxes or drop down boxes with only the valid values available.

Successful completion of the front end results in the creation of a properly defined parameter file that is then passed as input to the Box Tree System.

STILL TO DO

Install Front End Program in a Production Environment

One advantage of Java is that it is extremely portable across computing environments. This must be verified by running the front end program in the production UNIX environment. In addition, a script must be developed (this is not a difficult task) which will tie the front end to the existing Box Tree script to create a single run unit. A method must also be developed for this script to trap a return code from the front end program in case the front end is terminated without the desire to enter Box Tree in order to stop the script at that point. This, again, should not be a difficult task.

Implementation of Code Control

Some standards of version control need to be implemented to prevent multiple versions of the Box Tree System source from being used. As ongoing debugging efforts continue with this software, having multiple versions may result in extended life for some of these bugs. The need that different users of the software may have for modifications needs to be identified so that a system for organized code modification can be implemented, and perhaps a system of "user exits" needs to be placed in the code to allow for individual code modification.

Setting of default parameter values

The front end program has a default set of parameter values that is used for initialization if the "Use defaults" option is selected. This selection opens up the entry program to edit all the parameters. While this option may not be used very often (indeed, it is a primary purpose of the front end program to NOT have to do this), a set of base values for the default scenario needs to be specified and entered into the front end source code.

Creation of Runtime Scenarios for Parameter Settings

The aforementioned scenarios for various runtimes need to be defined and base parameter values specified. When this is done the front end program must be modified to reflect these scenarios.

Move parameter editing into the front end program

Parameter fields that are not set via checkboxes or drop down boxes are not edited for validity in the front end program. This means that illegal values will pass through to Box Tree and be caught via the edit routine there. It would be desirable to move most if not all (legality of certain values may be able to be determined only within Box Tree due to some internally set values) editing to the front end program so that errors may be caught and corrected without an aborted Box Tree run.

Removal of abbreviations

Abbreviations have been used throughout the code and in the parameter identifiers. Often this inhibits readability and can cause confusion (for example, does the abbreviation "mom" refer to momentum, moment, or mother; probably not mother). Since the parameter identifiers are now under the cover due to the new front end program, the change needed there is solely in the onscreen labels for those parameters, which are currently identical to the parameter identifiers. Confusing abbreviations in the code, however, need to be identified and changed using the mass change function of the code editor.

Investigate Removal of Pointers

One of the remaining issues regarding program readability is the extensive use of pointers throughout the Box Tree System. A pointer is a data item that contains a memory address for a data value rather than the data value itself. It can be useful at times in C programs and is essential when passing a parameter to a function that needs to be changed within the function (because of the "pass by value" nature of C functions). The

Box Tree System contains extensive use of pointers, however, that are referencing data structures that are defined and used within the current code block itself. This means that readability is affected because the pointer name is typically not descriptive of the data item to which it points. Since these pointers are used almost exclusively to point to data items that are data structures, they should be able to be replaced with the actual data names without affecting the run time of the software, since data names of data items which are data structures are themselves pointers. However, since executions of the Box Tree System can have very long run times, a change of this type must be accompanied by some extensive run time comparisons. Perhaps this change will be rendered unnecessary by the other data name changes already made, as this should help identify data items even though they are referenced (at the data structure level) by a pointer. Another consideration might be to change the pointer data names to be more descriptive of their use.

Other Debugging Issues

The Box Tree code is riddled with comments related to bugs or uncertainties concerning correctness or motivation of certain algorithms, and system users frequently encounter correctness issues. These issues need to be tabulated and prioritized to facilitate future debugging.

REFERENCES

Richardson, Derek C., Planetesimal Dynamics; download available on the internet at www.astro.umd.edu/~dcr/Research/thesis.ps.gz. Further information about the author of the code is available at www.astro.umd.edu/~dcr. This paper is in PostScript format which requires a reader. Some versions of Adobe Acrobat Reader will handle the PostScript format. A reader from Rampart Logic that seems to work well is available at download.cnet.com/Postscript-Viewer/3000-2094_4-10845650.html

java.sun.com – The Sun Microsystems website has everything needed to establish a development and runtime environment for the Java Virtual Machine

www.jcreator.com – The Xinox software website that contains free downloads for JCreator, a professional quality IDE for Java program development.

faculty.cs.wwu.edu/martin/software%20packages/BreezySwing/Default.htm – The website for downloads and documentation for BreezySwing, the software which provides for simplified implantation of GUI controls in Java programs.