

CONF-9606191--4

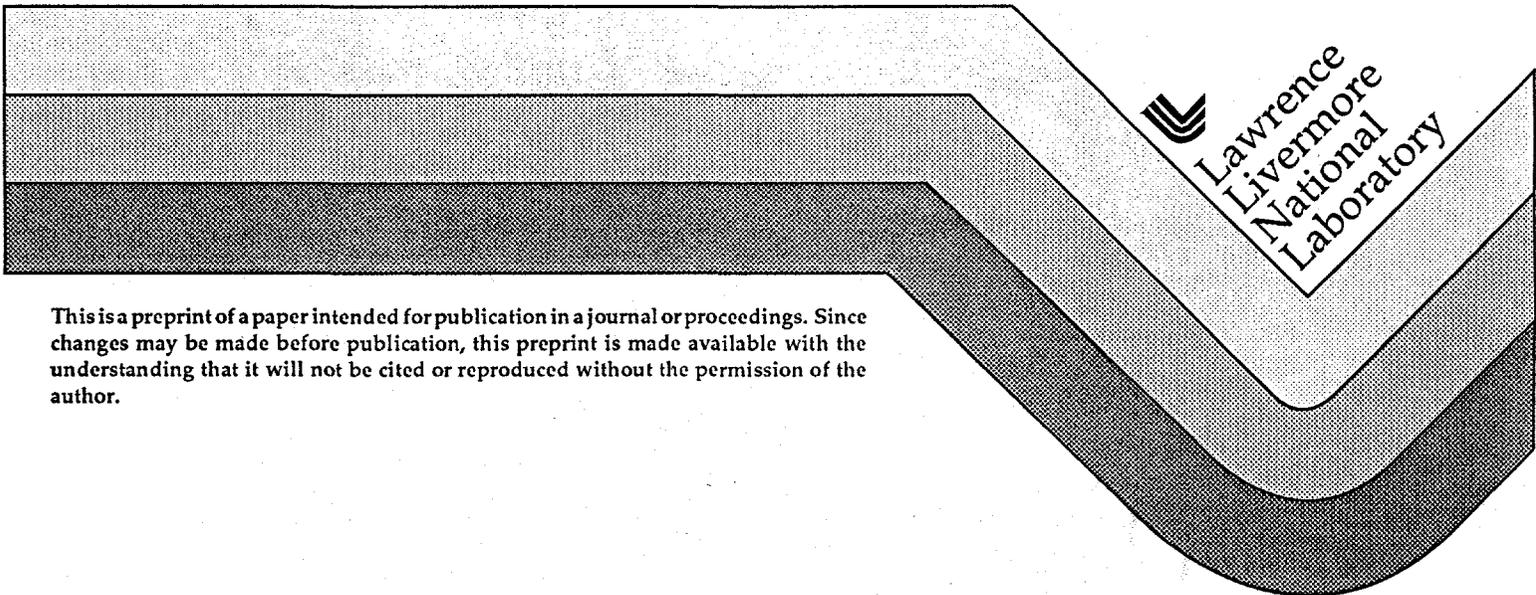
A Python Interface with Narcisse Graphics

Z. C. Motteler

RECEIVED
JUL 01 1996
OSTI

This paper was prepared for submittal to the
4th International Python Workshop
Livermore, CA
June 3-6, 1996

April 15, 1996



 Lawrence
Livermore
National
Laboratory

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

MB

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

A Python Interface with Narcisse Graphics

Zane C. Motteler¹

Narcisse

Narcisse is a graphics package developed by our French colleagues at Centre d'Etudes de Limeil Valenton of the Commissariat d'Energie Atomique. Narcisse is quite comprehensive; it can do two-, three-, and four-dimensional plots (the latter meaning that the surface is colored according to the values of an arbitrary function). One can open and send plots to a Narcisse window on a distant machine (handy if you want to show the boss a nice plot but you don't want to run over to her building with a hard copy).

Narcisse has a user-friendly graphical user interface (GUI) which, once a graph has appeared, allows the user to change its characteristics interactively. This enables one to find the best appearance for a particular plot without having to graph it repeatedly from the user program. Previously created files in various formats can also be imported directly into the Narcisse GUI and manipulated from there.

Unlike many graphics systems, whose functions are actually linked into one's program, Narcisse runs independently, as a graphics server. The user program communicates with Narcisse via Unix sockets. This communication is quite low level and very complex. The appearance of a plot is controlled by nearly 150 parameters for determining such things as the color palette, type of shading, axis scales, curve and surface labels, titles, angle and distance of view (for three- and four-dimensional graphs), hidden line removal, etc. Most end users do not wish to spend time learning the tedious details of such interfaces; they would just like to specify data and ask to have it plotted.

1. Zane Motteler is a computer scientist at Lawrence Livermore National Laboratory, P. O. Box 808, L-472, Livermore, CA 94551-9900. email: zcm@llnl.gov.

This paper describes a high level, easy to use graphics interface which hides (as much as possible) the low level details of whatever graphics system is actually being used, so that the low level can be essentially "plug-and-play." Then, whenever a better system becomes available, it should only be necessary to change low level interface routines not normally accessed by ordinary users. Python, with its easy extendability, was ideally suited for this job.

graphics_objects.py: Curve, Surface, and Mesh objects

Curve, Surface, and Mesh are three Python classes which abstract the geometric properties of the objects which we wish to graph. Instantiations of these objects will contain the geometric specification of one or more curves, surfaces, or meshes (coordinates and perhaps other data), as well as information about appearance (color, graph type, plotting options, etc.). Thus member functions will allow a curve, surface, or mesh to be added to or deleted from an existing object, and will allow appearance options to be changed for some or all of the geometric components. Another function allows the user to reinitialize an existing graphics object to be reused.

Finally, and most importantly, a Curve, Surface, or Mesh object can be asked to plot itself. The context in which a graph of a graphics object actually appears has been abstracted into a class called Plotter. Plotters know about such things as axes (whether they appear, what they look like, their scales, etc.), titles and other text which may appear on the graph, and (for three dimensional plots) the angle and distance of the view point. A Plotter is aware that a graphics engine is out there and knows how to interface with it. (See the next section for more details on Plotters.) A graphics object is aware that Plotters exist. Thus, when asked to plot itself, a graphics object is capable of instantiating its own Plotter, if one has not been supplied by the user. The end user of a graphics object does not need to know much about Plotters except that they need to be given information about a graph's context, as outlined above, and, of course, does not even need to instantiate one.

The end user who does not want a generic Plotter either can instantiate one or more Plotter objects and supply them to a graphics object, or can ask the graphics object to instantiate one or more Plotters, by specifying one or more character strings which somehow uniquely identify the Plotters. In the case of Narcisse, the character strings identify the server machine, the socket number, and the userid. Other

than the form which this character string must take, the user and the graphics object need have no further information about the structure or implementation of a Plotter.

Nar.py; the Plotter class

Nar.py, which embodies the definition of the Plotter class, is the low-level Python interface with the graphics engine, and as such is therefore acquainted with the graphics details. In our case a Plotter instantiation is aware of a connection to Narcisse, which is either handed to it by a graphics object or is created by default when it is instantiated. A Plotter has numerous somewhat higher level member functions which communicate with the lower level ones in the graphics interface. A Plotter object has two functionalities, namely:

- It operates as a conduit through which Curves, Surfaces, and Meshes can pass their geometric and graphical characteristics (color, label,...).
- It maintains information about aspects of the plot which are independent of the geometric abstractions being plotted, such as axis scales and limits, graph titles, existence or lack of a coordinate grid, etc., and for three dimensional plots, the angles and distance of the view point, etc.

One of the design decisions was exactly how to apportion the 150-some Narcisse parameters between Plotter objects and the various graphics objects. Generally I tried to keep all information intrinsic to a geometric abstraction (coordinates, color, appearance) with the graphics objects, and extrinsic information about the context of the plot (presence or absence of axes, point of view, size, particular Narcisse connection, etc.) within Plotter objects. Such a clean division was not always possible. For example, two dimensional plots in Narcisse can have their colors chosen from exactly one palette. However, three and four dimensional objects can be linked in such a way that each can have a separate palette. Thus in the former case, a palette could be considered a Plotter characteristic, while in the latter, it can be treated as a characteristic of an individual surface.

The settings of the most common of the low level Narcisse parameters are controlled by functions with self-explanatory names, such as `set_3d_grid_type`, `set_3d_options`, `set_axis_labels`, `set_curve_color`, `set_mask`, and even `set_language` (the GUI can appear either in French or in English). Since most graphics packages offer similar functionality, I hope that these names are sufficiently generic to be usable regardless of the underlying graphics. Curve plotting and surface plotting are each controlled by a single function (`plot_curve` and `plot_surface`, respectively).

This is possible because of the ability of Python to determine the number of arguments with which a function is called, and their types, sizes, and shapes. With this information, a Plotter can decide which of the several Narcisse functions is the appropriate one to call. Again, I hope that this will carry over to Plotter-type interfaces with other graphics packages.

narcissemodule

This module, written in C and callable (of course) from Python, is really a low-level set of wrappers for the functions in Narcisse. It accepts Python argument lists, parses them (if possible) into data acceptable to Narcisse, and passes it on. There is a local data structure for storing up Narcisse keywords and their values (which are used mainly to control the appearance of the graph) until they are to be shipped to Narcisse. Normally a whole series of keywords will be specified and then accumulated to configure a graph, but not sent to Narcisse until it is actually ready to do a plot. (If keywords were sent to Narcisse as they were defined, then whatever data that had been last sent would be replotted using the new values, which is usually not the desired outcome.) Finally, there is a local translation table: since Narcisse recognizes only French keywords, we have supplied a table of English equivalents to make things easier for the low-level user (mainly us, to be sure).

Thus there are very few surprises in this module. It contains routines to open and close a connection to Narcisse, to query whether one exists, and to synchronize. There are routines to add keyword values to the local table, and one to send the final list to Narcisse. Two dimensional routines can plot one or several curves simultaneously. The curves can be colored and labeled, and drawn in various ways (continuously, as step functions, or just as points marked by various symbols).

Surface plotting routines are where Narcisse excels. It supports both three dimensional plots, where the surface is colored according to the value of the z coordinate, and four-dimensional plots, where the coloring is determined by a function defined at each of the plotted points. There are various coloring options, including wire grid, flat or face coloring, contour lines, and contour shading. The surface can be transparent, or there are three masking options of increasing complexity to remove hidden portions from the graph.

Finally, there are routines for doing three dimensional solids which have been decomposed into cells by means of both structured and nonstructured grids. These

plots can be most enlightening when a portion of the surface is removed so that one can see the interior cells.

Examples of Plots

No paper on graphics is complete without a few sample graphs.

- Figure 1 shows a typical two dimensional graph. There is nothing spectacular here. We see that one can have multiple curves on the same graph, formatted and colored differently. The axes are shown with ticks and no grid.
- Figure 2 is a 3d surface plot in which an interesting surface mesh has been moved one unit (plus a small random perturbation) in the z direction and plotted a second time. 3d and 4d plots have four masking options: none (everything is transparent), minimum (surface is traced starting from closest part to view point), maximum (surface is traced starting from farthest part from view point), and sort (cells are sorted). A sort mask, the maximum, was necessary in order for these two surfaces to draw properly.
- In figure 3 we have a 4d plot of a sphere colored smoothly (the function whose values give the colors is random) and a 4d plot of a tetrahedron colored in "flat" mode, i. e., the faces are colored according to the average values of a function at the vertices. It is difficult in Narcisse to draw graphs containing multiple surfaces with different plotting options like this. It is necessary to set the axis maxima and minima by hand, set a "link" variable to 1 for the first surface and 0 for the subsequent ones, draw the most distant surface first, and suppress printing of the axes for all but the first.
- Figure 4 is a 4d structured mesh graph of the output of a three dimensional filament code. It is drawn with no masking and contour lines only. Only energies more than 7.5 times the initial value are shown.
- Figure 5 is a 4d unstructured mesh graph of an imploding sphere, with contours smoothly colored according to the z velocity component. The front half (more or less) of the cells have been stripped away to show the interior velocity distribution.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

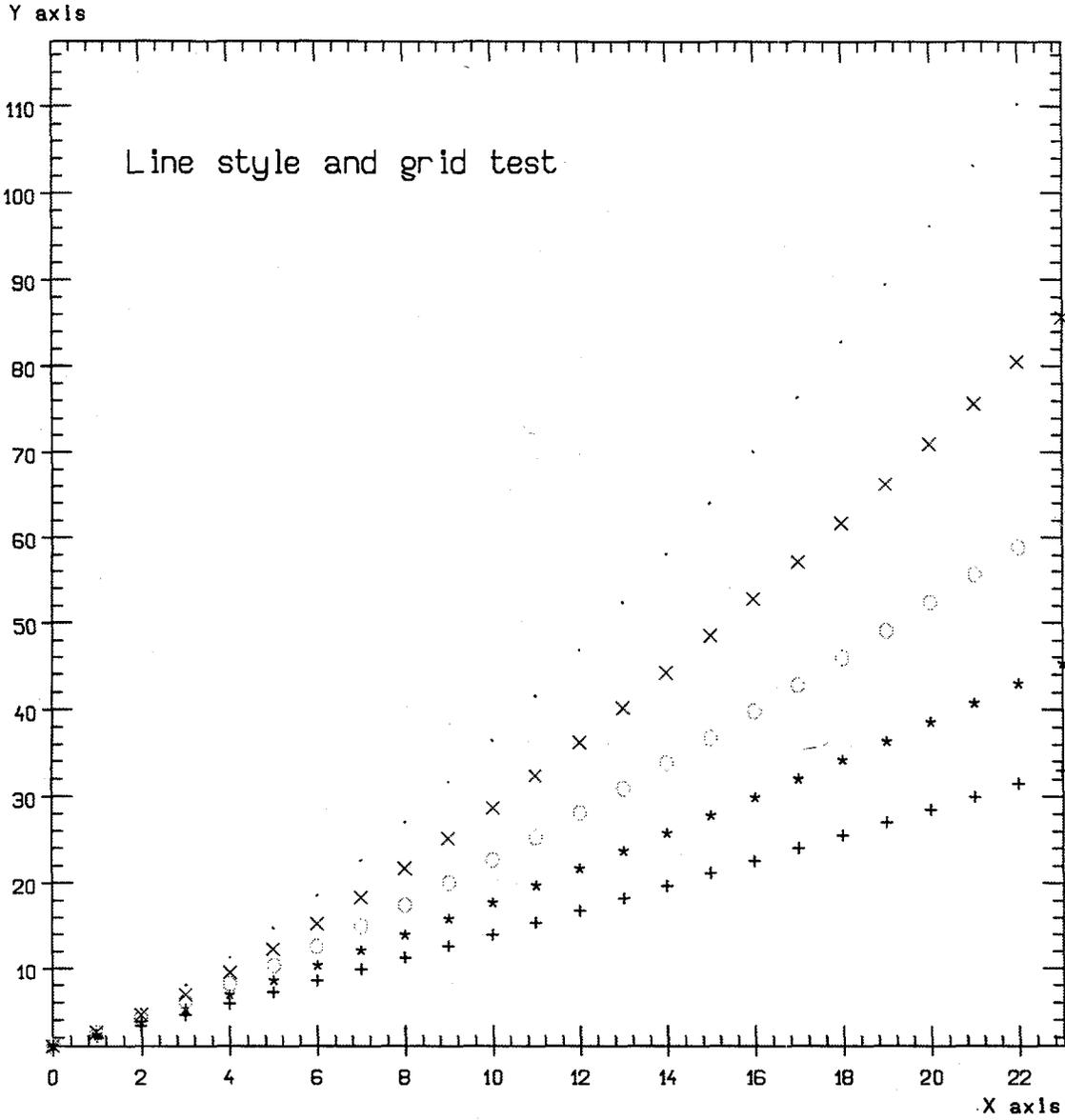
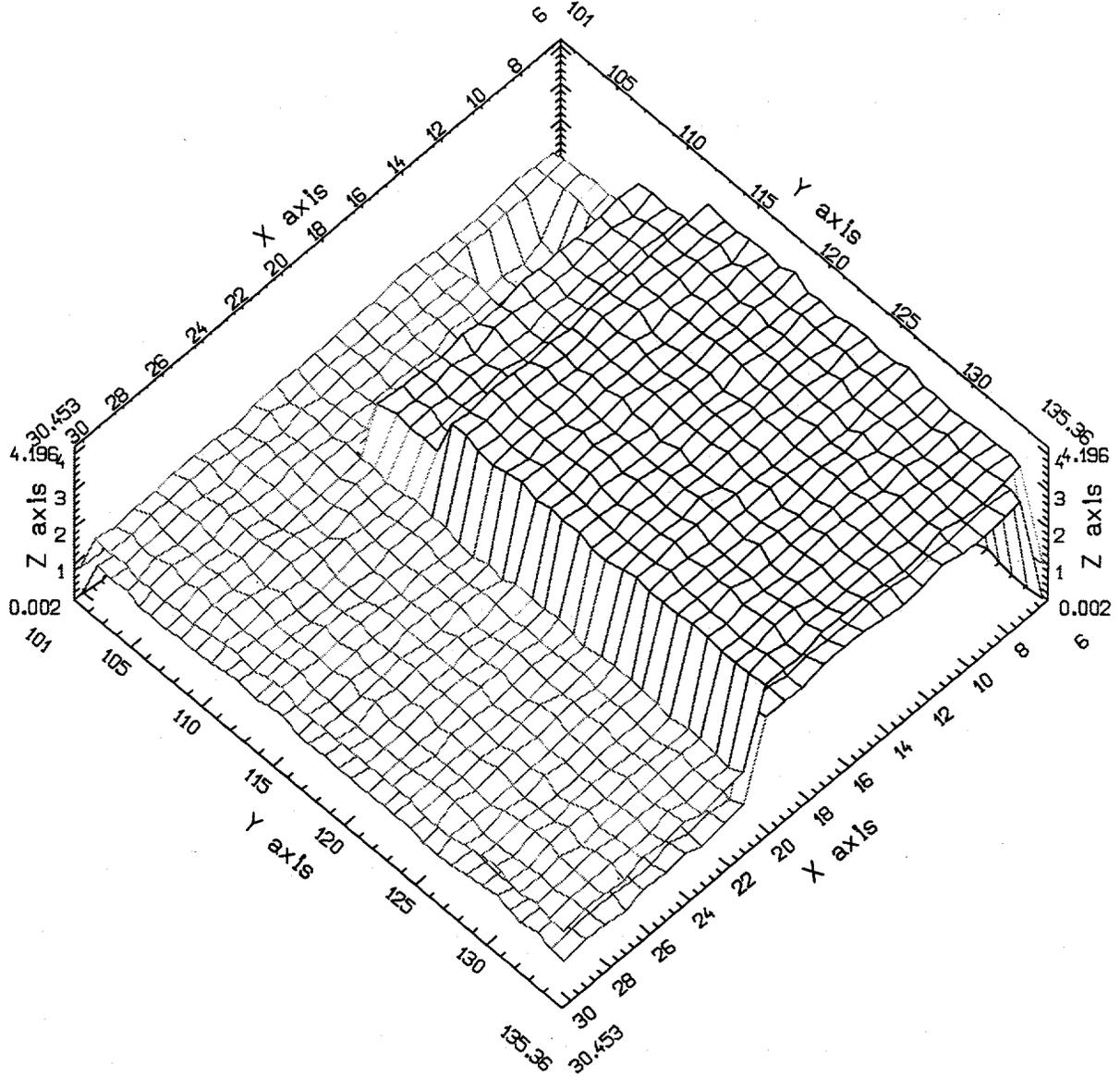


Figure 1

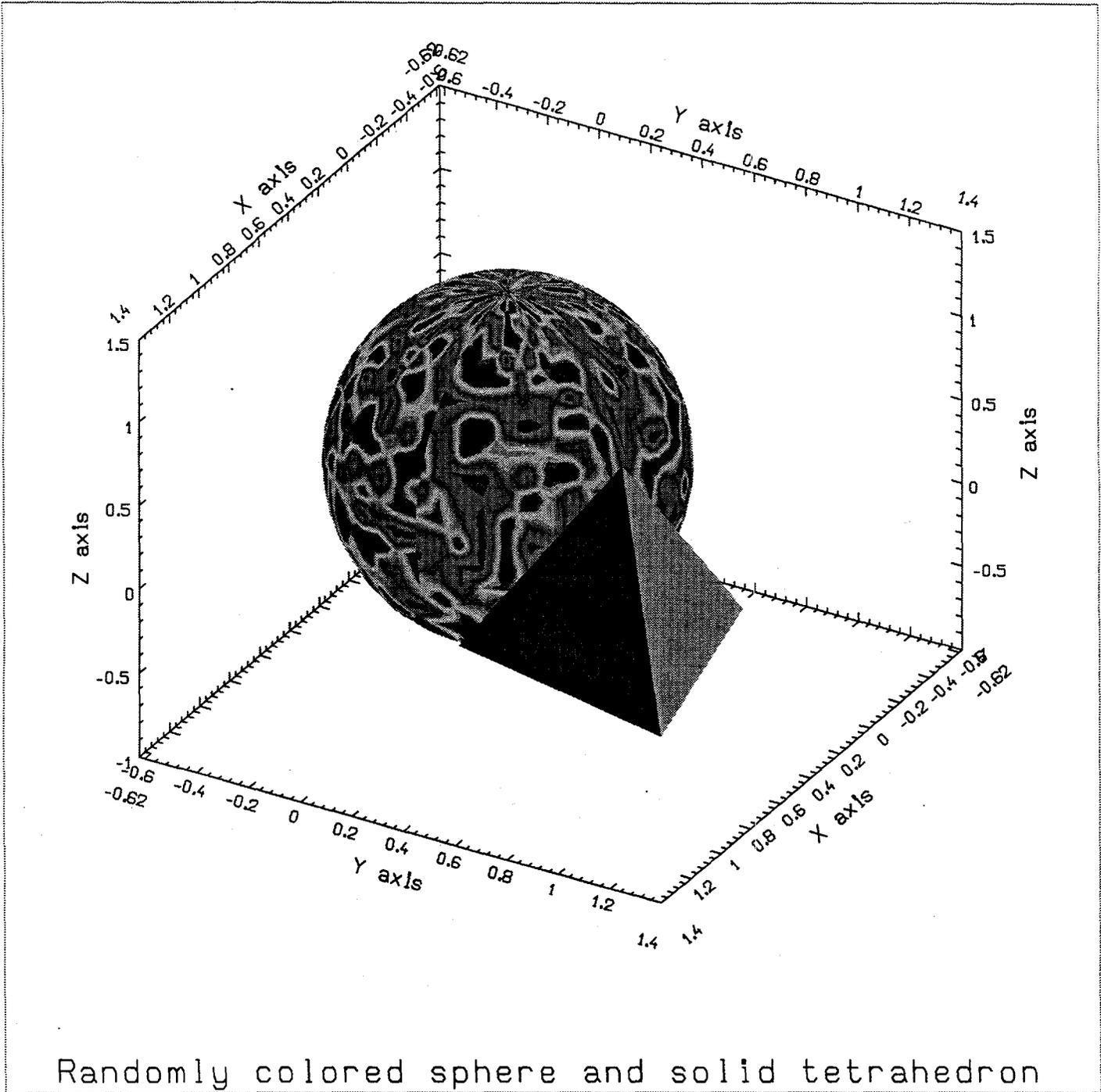


(Animation)



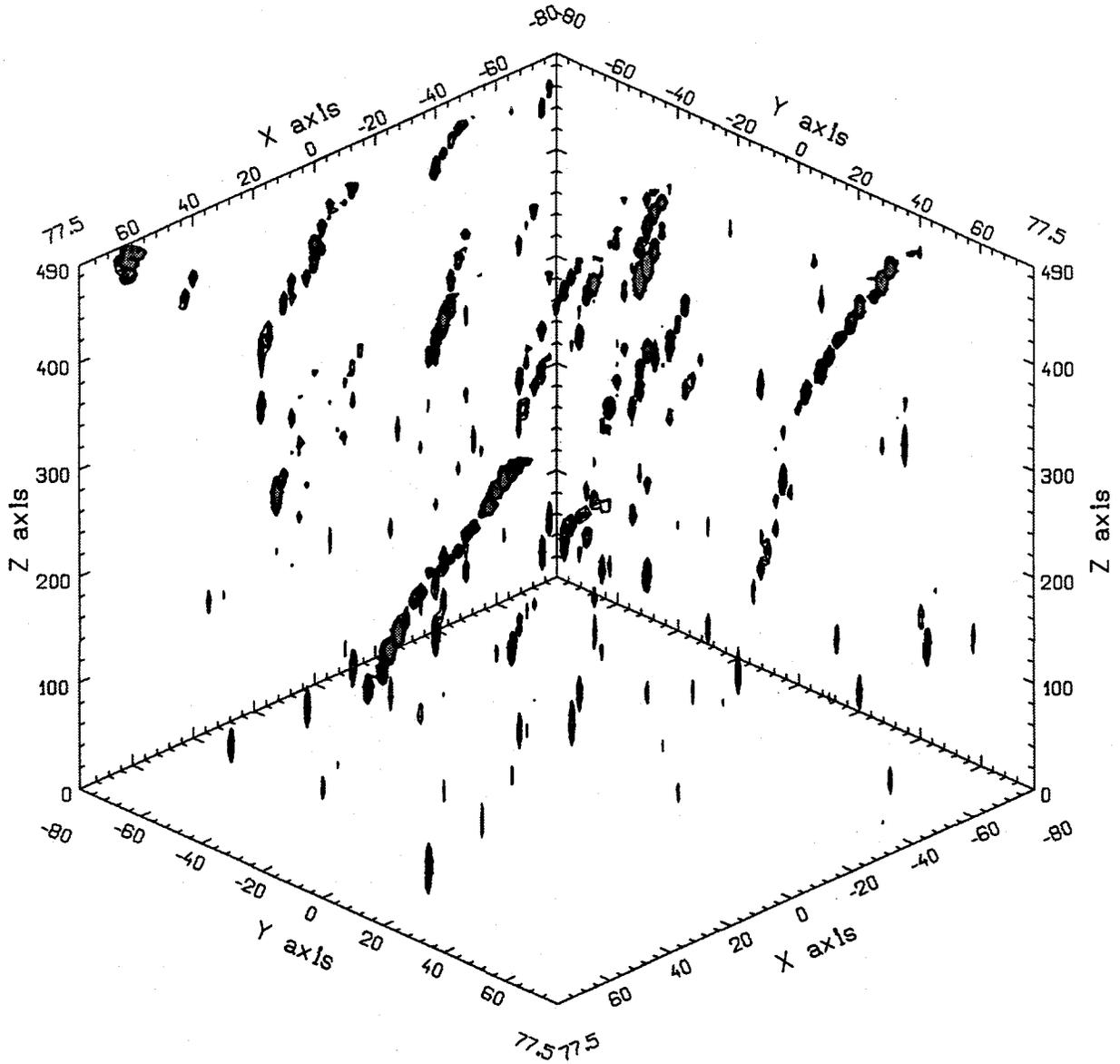
Mesh follows velocity field one time step

Fichier : /home/cs/motteler/figure2.ps level 2
Utilisateur : motteler Le : 12/04/1996 a 08:26:30 Casier : Rm 1329
Temps d'impression: 0h 0m 27s



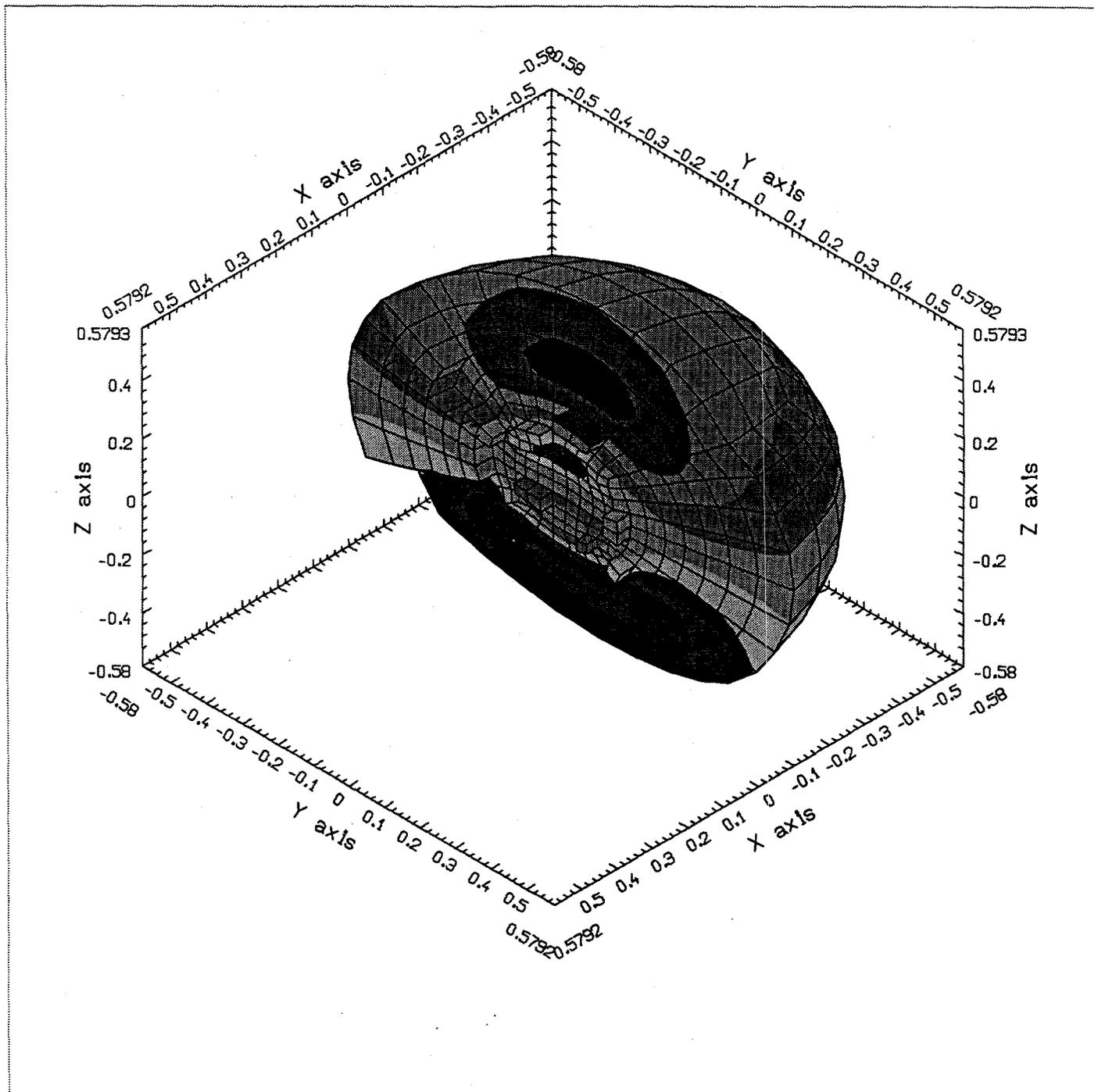
Randomly colored sphere and solid tetrahedron

Fichier : /home/cs/motteler/figure3.ps level 2
Utilisateur : motteler Le : 12/04/1996 a 08:59:11 Casier : Rm 1329
Temps d'impression: 0h 2m 47s



Energy Filaments, cutoff = 7.5

Fichier : /home/cs/motteler/figure4.ps level 2
Utilisateur : motteler Le : 12/04/1996 a 09:13:15 Casier : Rm 1329
Temps d'impression: 0h 1m 6s



Fichier : /home/cs/motteler/figure5.ps level 2
Utilisateur : motteler Le : 12/04/1996 a 09:50:11 Casier : Rm 1329
Temps d'impression: 0h 3m 24s