1 |IZIC 1.0 : An Overview for the User

Norbert Kajler[1]
RIACA
Kruislaan 419, 1098 VA Amsterdam
E-mail: kajler@can.nl

Published in: Computer Algebra Nederland, ISSN 1380-1260, issue 13, pages 17–25, Dec. 94.

1.1 Introduction

Graphing packages are included for free with most commercially available Computer Algebra Systems (CAS). For instance, Axiom, Macsyma, Maple, Mathematica, and Theorist all provide 2D and 3D plotting with many options to change colors, labels, viewpoint, or painting style. In addition several of them allow interactive rotating, choice of lighting model, or recording of frames for later replay and animation.

However, when compared with state of the art graphics applications, these graphing packages are still primitive. Actually, implementing an efficient visualization software is a hard task which goes far beyond the usual scope of any CAS. This is why several CAS have recently included links to commercial high-quality scientific visualization packages such as AVS [UFK+89] and IRIS Explorer [Edw92]. This way, graphics objects are still computed by the CAS, but the resulting data is exported to an external renderer which can provide much better visualization quality and additional control on the image produced. These packages are usually implemented on top of some 3D libraries such as PEX or OPEN-GL which are designed to efficiently manipulate 3D objects on high-end UNIX workstations with optimized hardware. An inconvenience of this approach is that it requires additional expensive software (i.e. AVS or IRIS Explorer) which themselves rely on specific software and/or hardware layers. For researchers or advanced users, another possible inconvenience is that all the software involved is provided as compiled code which can hardly be altered.

The purpose of IZIC is to provide a simple but still powerful surface visualization package especially designed for display and interactive manipulation of 3D mathematical objects. An additional design characteristic is that IZIC does not rely on any proprietary software layer beyond Unix. It is therefore usable on most Unix workstation running X11. IZIC is also independent of any particular CAS. It runs as an independent process and comes with four separate files to interface it with Macsyma, Maple, Mathematica, and Reduce.

IZIC was designed in 1992-1994 at INRIA Sophia-Antipolis and RIACA by Robert Fournier, Norbert Kajler, and Bernard Mourrain, with major contributions by Guillaume Fau. Additional contributions include the Macsyma and Reduce interfaces implemented respectively by Olaf Bachmann and and Winfried Neun. A preliminary version of the software was demonstrated for the first time during the DISCO'93 conference in Gmunden, Austria. IZIC 1.0 is now freely available via anonymous ftp from the server `zenon.inria.fr` in the directory `safir` [2]. The archive file includes the interfaces for Maple, Mathematica, and Reduce. The interface for Macsyma should be available soon.

[1] Also: Projet SAFIR, CNRS/INRIA/I3S, Université de Nice-Sophia Antipolis, Bat. 4, 250 rue Albert Einstein, Sophia-Antipolis, F-06560 Valbonne, France, e-mail: kajler@mimosa.unice.fr

[2] There is also an electronic exhibition of mathematical pictures done with IZIC which is freely accessible with xmosaic at the following address: http://www.inria.fr/safir/SAM/Izic/images.html.

1.2 Computing Mathematical Objects

As stated previously, one can use IZIC from Macsyma, Maple, Mathematica, or Reduce. One can also connect IZIC to a different system by implementing a new interface. As an example, we will present in this section how to use IZIC from Maple using the mapleizic interface. Interfaces for others CAS are very similar to mapleizic, functions names and signatures being even the same in most cases.

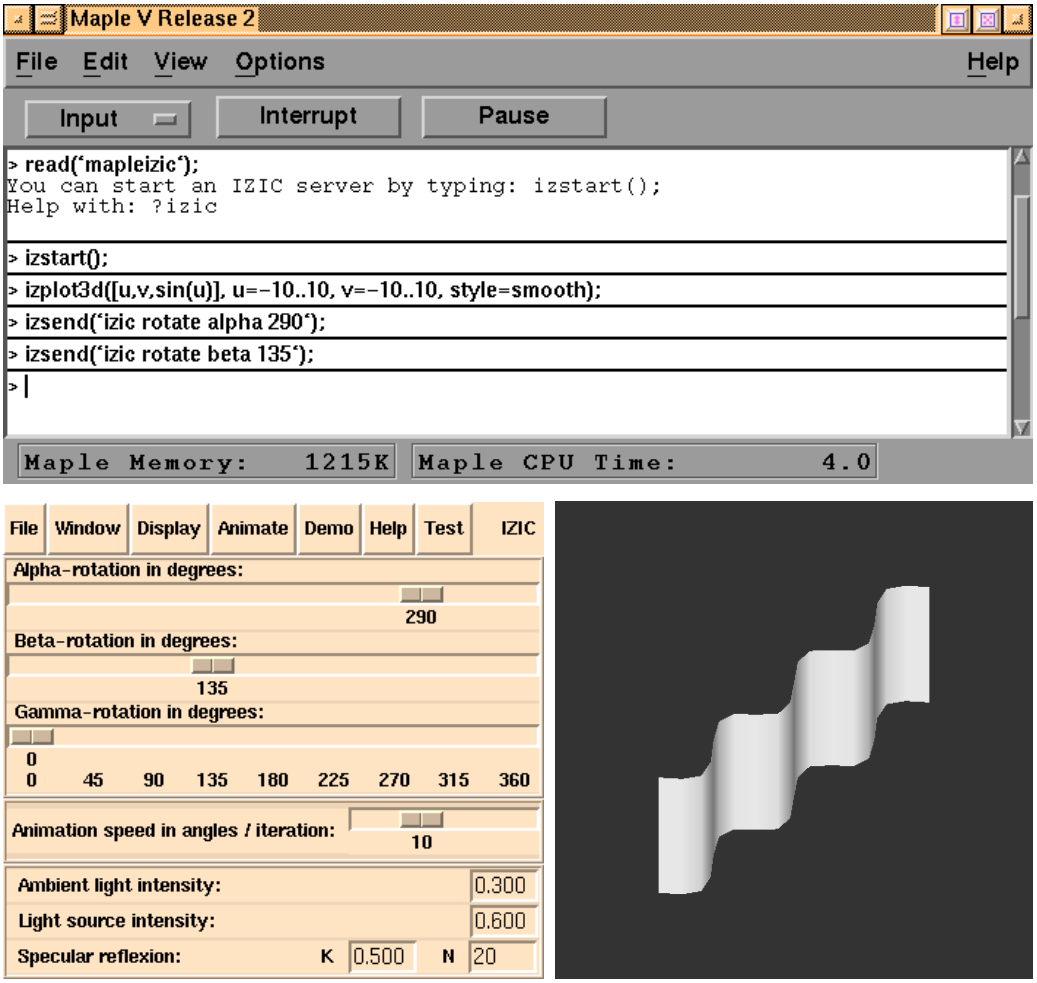

Figure 1.1: Using IZIC from Maple

Using IZIC from Maple requires first to load the `mapleizic` file. This can be done by typing the command `read(mapleizic);` or by inserting this command once for all in your `.mapleinit` file. If the IZIC program is not installed in a standard location you may have to change the very first lines of mapleizic. Once mapleizic is loaded, a message is displayed to remind the user that an IZIC server should be started next if not done already, and that an on-line help on IZIC is available by typing `?izic`. Also, loading mapleizic defines several new Maple functions allowing computation of curves and surfaces and remote control of IZIC from Maple (see Table 1.1). As a first example, Figure 1.1 presents the beginning of a Maple session in which we successively:

1. load the mapleizic interface file;
2. start an IZIC server (which makes the IZIC control panel appear);
3. plot the surface z=sin(x) expressed with a parameterization of u and v (which makes the graphics window appear);
4. send two commands to IZIC asking for a rotation of 290 degrees along the x-axis, and 135 according to y-axis.

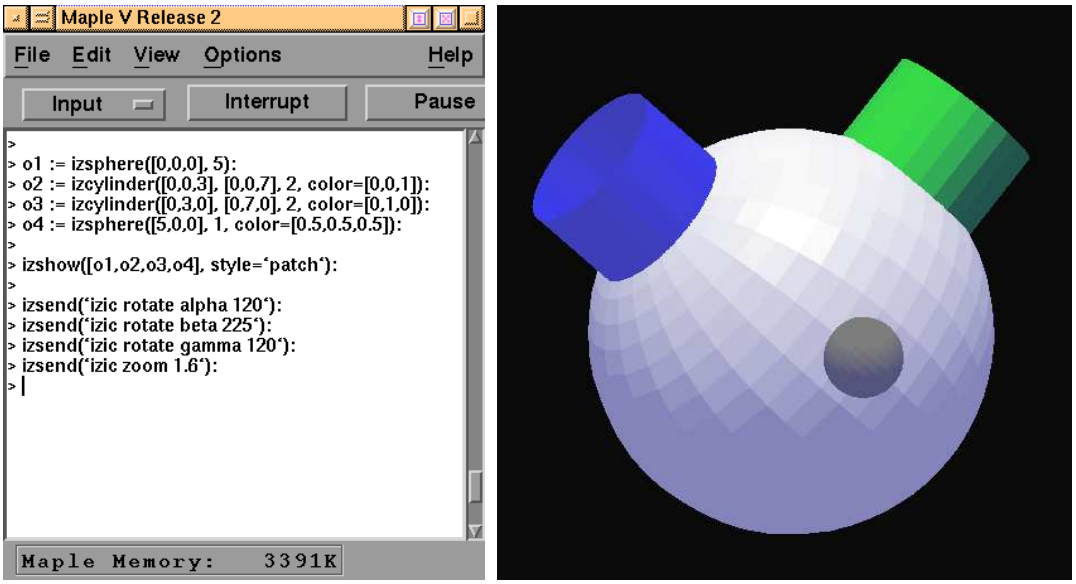

Figure 1.2: Displaying several objects together

The izplot3d function is actually included for convenience as it simply combines in one operation the computation of a parametric surface (purpose of the izsurface function) and its immediate display in an IZIC window (purpose of the izshow function).

Maple command	**purpose**
izstart()	Start a new IZIC server
izsend(*str*)	Send a TCL scipt to IZIC
izsetoptions(*option*)	Return default value for option
izsetoptions(*option* = *value*)	Change default value for option
izplot3d(*vector*, $range_1$, $range_2$, *options*)	Compute and visualize a parametric surface
izplot3d(*vector*)	
izcurve(*vector*, $range_1$, *options*)	Compute and return a parametric curve
izsurface(*vector*, $range_1$, $range_2$, *options*)	Compute and return a parametric surface
izline($point_A$, $point_B$)	Return a line
izcylinder($point_A$, $point_B$, *radius*)	Return a cylinder
izsphere($point_A$, *radius*)	Return a sphere
izshow(*obj*, *options*)	Visualize an object (curve or surface)
izshow([*list_obj*], *options*)	Visualize a list of objects (curve or surface)

Table 1.1: List of mapleizic functions

Figure 1.2 shows a second Maple session where several objects are first computed before being displayed together using izshow. Notice the use of izcylinder and izsphere which are two of the convenience functions provided by mapleizic to ease the creation of most elementary curves and surfaces.

Figure 1.3 shows that the output of the Maple function tubeplot can be used as input to izshow. Actually, izshow can display any graphic object computed by Maple using functions such as plot3d or tubeplot. The transparency option (which produces the right hand graphics) is one of the various IZIC options which can be set from Maple via izshow. This example is taken from the Maple V Language Reference Manual where the same equations are used to compute the surface shown on Plate 6.

As already mentioned, a collection of options with customizable default values allow to customize colors, transparency, display style, etc. Most of these parameters can also be changed interactively from the control panel. Table 1.2 gives the list of possible options together with their default values. These options can be used with most mapleizic commands related to object computing or display, such as izplot3d or izshow. Default values can also be changed using the izsetoptions command. For

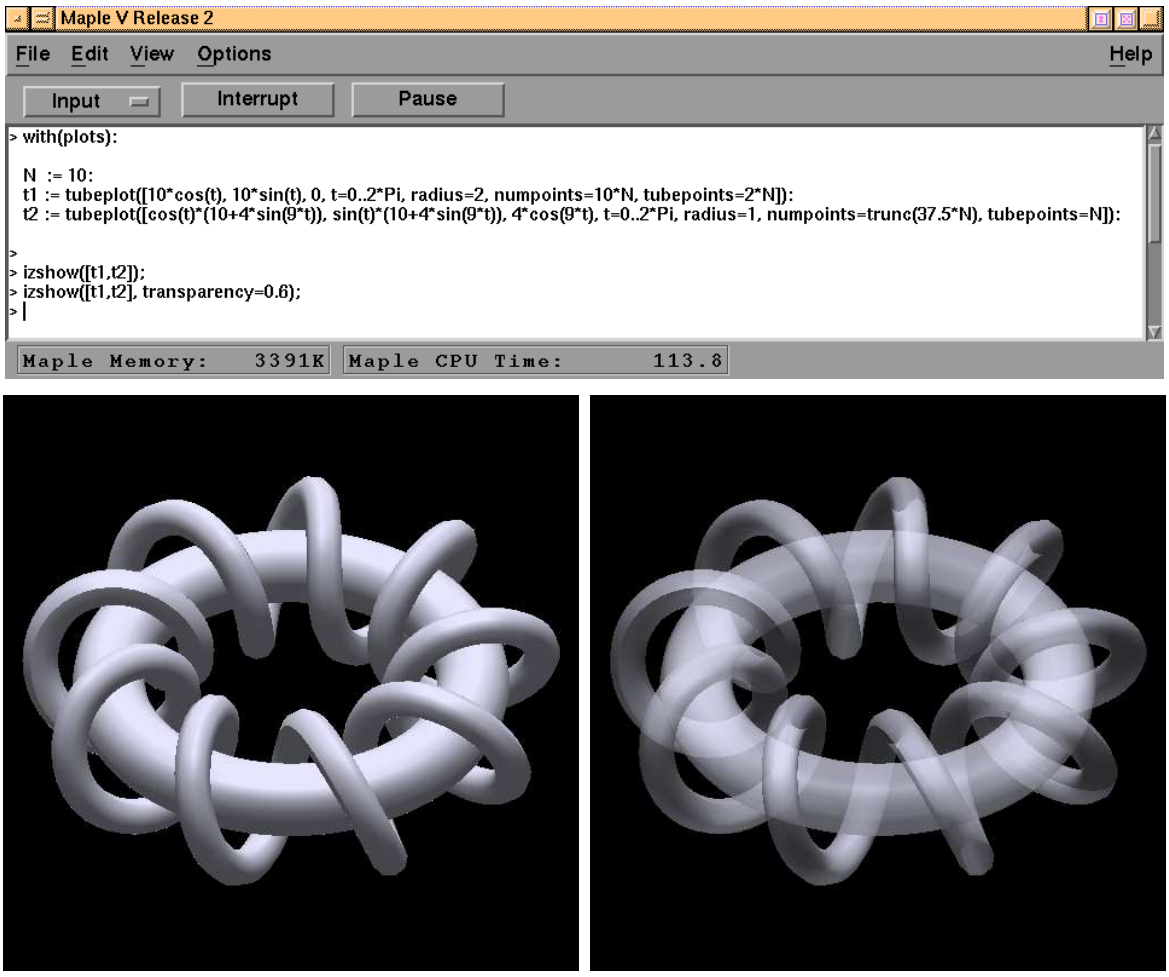

Figure 1.3: Displaying Maple's tubeplot function output via IZIC

instance, `izsetoptions(mincolor=[0.15, 0, 0], maxcolor=[1, 0, 0]);` will set default values so that surfaces will be displayed using a color gradation from nearly black to pure red[3]. Among

Option	default	purpose
box	[-5, 5, -5, 5, -5, 5]	specifies the bounding box
fgcolor	[0.0, 0.0, 0..0]	specifies the foreground color
bgcolor	[1.0, 1.0, 1.0]	specifies the background color
curvecolor	[0.0, 0.0, 1.0]	specifies the curve color
mincolor	[0.1, 0.1, 0.1]	specifies the lower value in color range for surfaces
maxcolor	[1.0, 0.0, 0.0]	specifies the upper value in color range for surfaces
padding	0.0	specifies the quantity of space added around the object
style	wireframe	specifies how the surface is to be rendered
range	-10..10	specifies the range of each variable in a surface
curvegrid	[200]	specifies how many points are computed when plotting a curve
grid	[25,25]	specifies how many points are computed for each variable
transparency	0	specifies transparency of the surface

Table 1.2: List of mapleizic options

most useful options, "style" specifies how the graphics will be first displayed. Possible values include

[3]Colors in IZIC are specified according to the RGB model: three floating point numbers (all in the range 0..1) indicate the mixture of Red, Green, and Blue.

wireframe, hidden, patch, smooth, and auto. The value "auto" is a special one which will let IZIC automatically select the best available display style (according to the object's complexity) so that the object can still be animated in real time.

1.3 Visualizing curves and surfaces with IZIC

Once computed by a CAS, curves and surfaces can be displayed according to several modes and manipulated interactively.

1.3.1 Reading ZIC data files

The `File` menu of the control panel allows to read ZIC files previously generated by Maple (or any other CAS interfaced with IZIC). This is useful for browsing collections of pre-computed objects: when using IZIC from a CAS, the interface manages transparently the creation, sending, and display of ZIC data files. For demonstration purpose, a collection of pre-computed objects are also provided with IZIC 1.0 and can be loaded from the `Demo` menu.

1.3.2 X and Z window

IZIC can display a surface either as a wireframe object in a monochrome window (called X window); or as a collection of polygons in a colored window (called Z window). Several display options are available in both cases in order to perform hidden parts removal, smooth shading, etc. Clearly graphics in X windows are fast to compute and can be easily manipulated in real-time. Alternatively, to produce graphics in Z windows is much more computationally intensive. Furthermore, such graphics can be manipulated in real-time only on powerful enough workstations or in the case of relatively simple objects.

By default IZIC will display objects so that real-time manipulations should always be possible. An heuristic takes automatically care of the computer and display performances, resulting in a display in an X or Z window according to object complexity. Then, the `Window` menu allows to open and close alternative views in X or Z windows. If an object is too complex to be manipulated in real-time, it is convenient to rotate and scale it in the X window before opening the Z window and setting correct lighting.

1.3.3 Playing with four views

In order to give a better insight of 3D objects, IZIC manages four views of the displayed curves and surfaces. When a new object is displayed, the four views are initialized such as the views #1 and #2 show the objects from the front, #3 is a view from the right side and view #4 from the top. By default, only the first view is displayed and the `Display` menu allows to switch views or to show all four views together in the same window. Figure 1.4 shows four views of the same moebius band in both X and Z windows.

1.3.4 Transparency

A nice feature of the ZIC library is its support for transparent objects. Each object as its transparency parameter set to 0 by default, resulting in an opaque object. When setting transparency to say 0.5, the object becomes semi-transparent which means that another object laying inside it could be seen as in figure 1.3. However adding transparent objects makes display dramatically slower due to the use of more computationally intensive algorithms.

1.3.5 Display options

The `Display` menu of the control panel allows to switch several parameters related to the rendering process.

For an X window, only one parameter can be switched: the **`painter`** entry which controls whether hidden lines should be removed from the wireframe picture or not. Clearly, using the painter algorithm

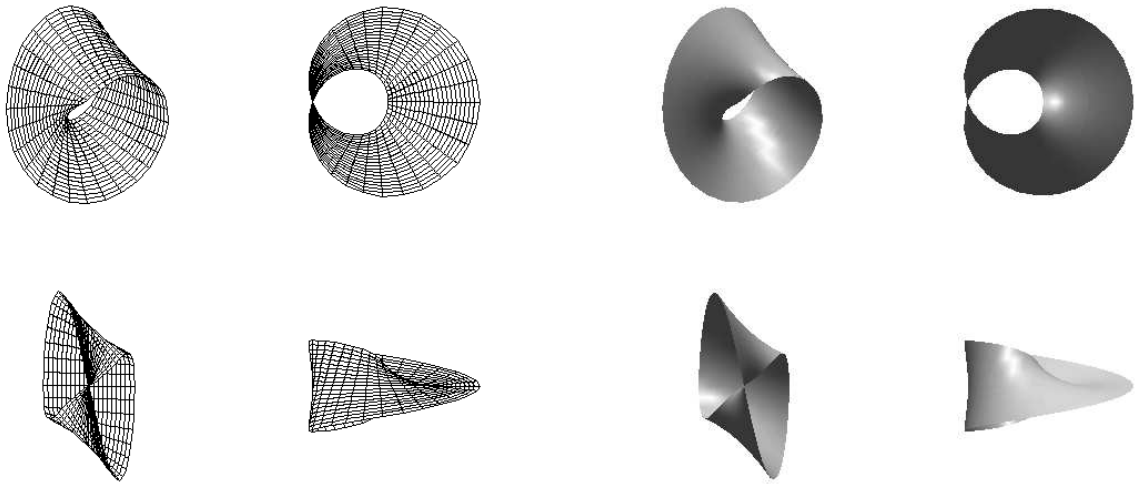

Figure 1.4: Four views of a Moebius band in wireframe and smooth sading modes

makes wireframes figures much easier to understand, but may result in slower display on low-end workstations.

For a Z window, several options are available. Figure 1.5 presents the same amonite object displayed according to the following options: wireframe (a); wireframe with hidden lines removed (b); shaded polygons (c); smooth shading (d); smooth shading with different light source setting (e-h); smooth shading with different specular reflexion setting (i-l); smooth shading and outside meshed (m), non lighted (n), hidden (o).

1.3.6 Interacting with the mouse and keyboard

Scaling, moving, and rotating objects in a graphics window can be performed by direct manipulation of the graphics with the mouse. Each action is bound to a mouse motion with left, midle, or right button pressed respectively. Several key-bindings are also provided as accelerator for various commands available from the control panel. The `Help` menu includes an entry to display all mouse and keyboard bindings.

1.3.7 Interacting via scripts

All functionalities available from the control panel or via the mouse are actually implemented as TCL commands. At any time, IZIC can receive a TCL script sent remotely from a CAS. It can also interactively interpret TCL scripts entered via the command line editor of the control panel. In addition, at startup time IZIC always looks for a file named `.izicrc` in the user's HOME directory. If such a file exists, it is interpreted as a TCL startup script, allowing customization of IZIC by end users.

1.3.8 Animation

As an application of writing TCL scripts, different kind of predefined animations are available from the `Animate` menu. This includes spinning around each axis and zooming/un-zooming. By embedding a scripting language providing loops, users can create more sophisticated animations by controlling simultaneously rotations, scaling, and all lighting parameters.

1.3.9 Generating postscript from IZIC windows

The `File` menu includes an entry to save the mesh visualized in the X window as a postscript file. No commands are currently available to translate the Z window content into postscript. Examples

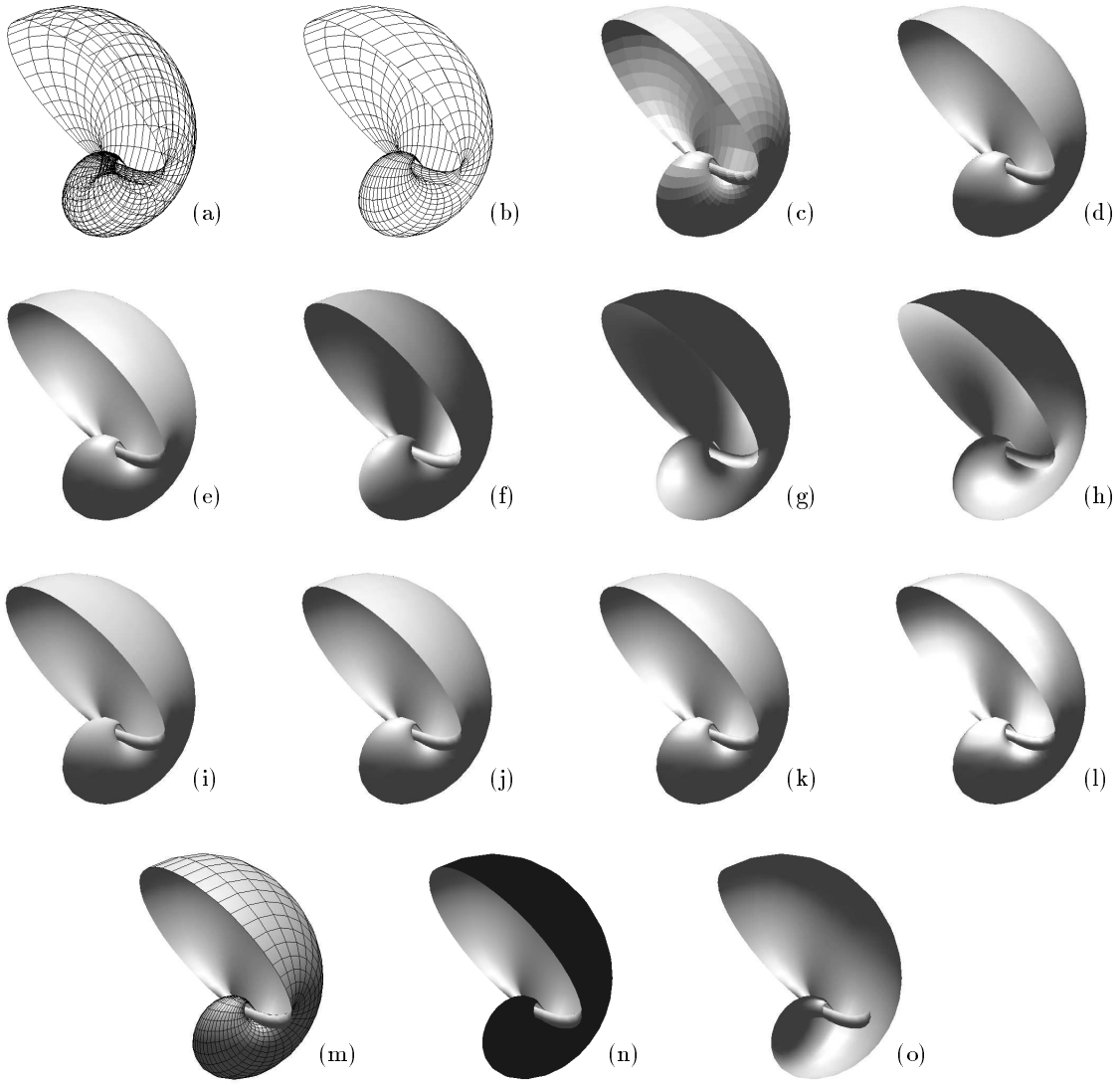

Figure 1.5: Display modes

presented in this paper were produced using standard X11 screen capture programs (which means that what is shown on the paper has the same resolution as what was displayed on the screen).

1.4 Architecture and Implementation

Details on the implementation of IZIC can be found in [FKM93] and [Fou92]. In short, the architecture of IZIC 1.0 consists of ZICLIB [Fou92], a hardware-independent C library providing a large collection of graphical operations, linked with a TCL interpreter [Ous94]. At runtime, IZIC can be driven both through its graphic control panel and via TCL scripts which can be sent remotely from one or more Computer Algebra Systems. With the present implementation of IZIC, graphic objects to be visualized are defined using an ASCII representation: the ZIC format. Files in this format include a series of optional and mandatory fields: type of the objects, bounding box, colors, object encoding, etc. Typically, *object encoding* is a long collection of floats which structure depends upon the type of the object.

The role of an interface between a CAS and IZIC is therefore to generate representations of curves and surfaces in this format, and then, communicate with IZIC by sending TCL scripts in order to

remotely animate the surface, change the lighting parameters, etc. Typically such an interface provides a collection of user-level commands (such as izplot3d, izcurve, izsurface, izshow, izsend, izline, etc.) which extend or supplement existing graphing capabilities of the CAS. Advanced users can easily extend an existing CAS interface such as mapleizic, or create a new interface for their own system. Mapleizic for instance is implemented as 950 lines of Maple code, including the on-line help pages.

Clearly, by totally decoupling the computation of mathematical objects and their visualization, end users can select the CAS they prefer to construct the data, and then use IZIC (or any other equivalent software) to display and manipulate the graphics.

1.5 Limitations

Several useful features are clearly missing in the current version of IZIC. Some of them should be added with the next release: axis, interactive selection of objects when a surface is made from several pieces, incremental addition of new objects, etc. Several desirable extensions require more work on the ZIC library. This includes adding high-quality output for printing or supporting always more sophisticated lighting models. Also, major improvements could be achieved by setting up more efficient communication mechanisms between CAS and IZIC. In the current version, data are sent in an ASCII format, using files. Two major improvements would be to use sockets instead of files, and even more importantly, to avoid translating floats from the CAS side into an ASCII representation which is immediately parsed on the IZIC side to generate floats again. Clearly, the transfer of numerical data between the two processes would be much more efficient by sending binary data via a socket. Unfortunately this is simply impossible to implement with most CAS at this time because of the lack of routines for opening a socket or outputing floats in a binary form.

Bibliography

[Edw92] G. Edwards. *Image Processing*, chapter Visualization – the Second Generation, pages 48–53. 1992.

[FKM93] R. Fournier, N. Kajler, and B. Mourrain. IZIC: a Portable Language-Driven Tool for Mathematical Surfaces Visualization. In A. Miola, editor, *Proc. of DISCO'93*, pages 341–353, Gmunden, Austria, September 1993. LNCS 722, Springer-Verlag.

[Fou92] R. Fournier. ZICVIS et la ZIClib – Visualisation de maillages de surfaces. Version 1.6. INRIA, Sophia Antipolis. French INRIA internal report.

[Ous94] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[UFK+89] C. Upson, T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. The Application Visualization System: A Computational Environment for Scientific Visualization. *IEEE Computer Graphics and Application*, 9(4):30–42, July 1989.