Visualizing Maple Plots with JavaViewLib

Steven Peter Dugaro[1] and Konrad Polthier[2]

[1] Center for Experimental and Constructive Mathematics, Simon Fraser University, Canada
[2] Institute of Mathematics, Technical University Berlin, Germany

Abstract. JavaViewLib is a new Maple package combined with the JavaView visualization toolkit that adds new interactivity to Maple plots in both web pages and worksheets. It provides a superior viewing environment to enhance plots in Maple by adding several features to plots' interactivity, such as mouse-controlled scaling, translation, rotation in 2d, 3d, and 4d, auto-view modes, animation, picking, material colors, texture and transparency. The arc-ball rotation makes geometry viewing smoother and less directionally constrained than in Maple. Furthermore, it offers geometric modeling features that allow plots to be manipulated and imported into a worksheet. Several commands are available to export Maple plots to interactive web pages while keeping interactivity. JavaViewLib is available as an official Maple Powertool.

1 Introduction

Application connectivity refers to one programs' ability to link to other programs. In the past, application connectivity has typically been of secondary importance to the Mathematics community. Software is commonly developed from the ground up to realize the research goal, not the potential for integration with other mathematics applications. However, some applications do make provisions and have great connectivity. JavaView [5][6] provides an api for 3rd party development in addition to great import and export utility for the exchange of geometric data. Mathematica [7] has provided a very thorough interface known as MathLink on top of which a Java version known as J/Link allows Java programs to be controlled from within Mathematica, and the Mathematica kernel to be controlled from within any Java program. In fact, the ease of application connectivity provided by these two applications has already allowed for their quick and seamless integration[1].

This paper documents the authors' efforts on JavaViewLib to establish connectivity between JavaView and Maple [3], unite their strengths, and extend their functionality. JavaView and Maple vary in scope, but it is clear that one application's strengths overlaps the other's shortcomings. While Maple is a powerful tool for algebraically obtaining and generating visualization data, JavaView is a superior geometry viewer and modeling package capable of displaying geometries dynamically in html pages.

[1] `http://www.javaview.de`

The JavaViewLib (JVL) is an amalgam of JavaView with a Maple library; an interface between them. It makes use of the strong aspects of both to facilitate the exchange of geometries between the two, quickly build web pages with those geometries, and enhance the experience one has with Maple plots. JVL is typically used to preserve the dynamic qualities of a Maple plot upon export to the web; static plot snapshots in Maple html exports can now be replaced with dynamic plots in a JavaView Applet. Via JavaView, geometries can be exported from Maple to a variety of modeling packages and vice versa. JVL can quickly build geometry galleries with little effort, and these geometries can be exported and displayed in a legible XML format to ease further development. JavaViewLib is available as an official Maple Powertool [4].

2 Visualization in Maple and JavaView

2.1 Graphics in Maple

Maple is a comprehensive computer algebra system. It offers packages from various branches of advanced mathematics to aid in solution and visualization. It also offers word processing facilities with typeset mathematics and export capabilities to aid in the composition of mathematical papers. Furthermore, it provides a unique environment for the rapid development of mathematical programs using its own programming language and function libraries. These features are encapsulated in a single Maple Worksheet (mws), which can be transported and rendered between Maple applications.

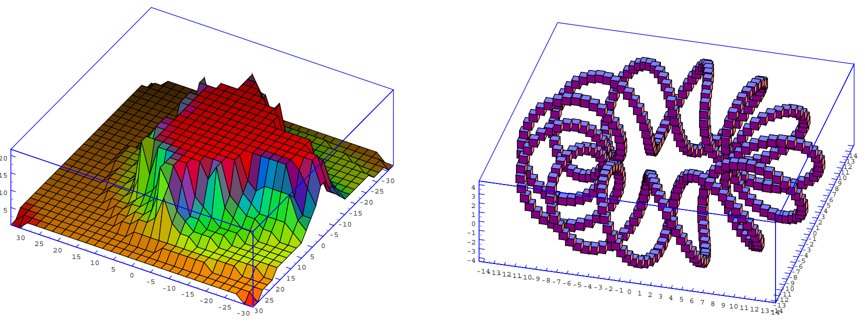


Fig. 1. JavaView parses the color of meshes and viewing options like axis and frames of Maple plots.

Maple's mathematics engine is among the industry's best, but there is room for improvement in certain areas of the application overall. While Maple supplies numerous methods for generating graphics from mathematical expressions, the viewer itself has a primitive feature set. Once the graphic –

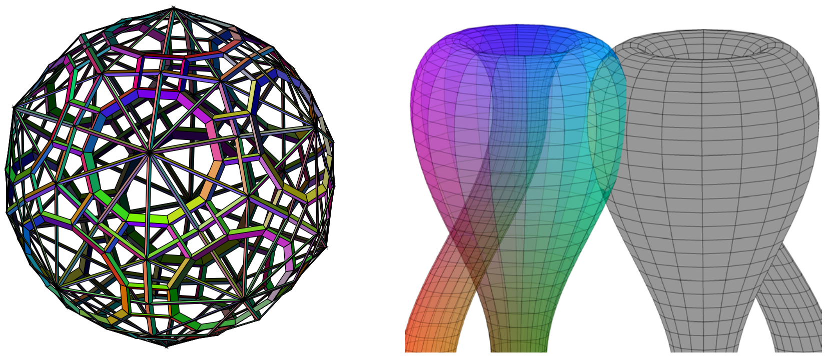


Fig. 2. Maple Plots with transparency visualized and colored in JavaView, for example, to show the hidden inner tube of the Klein bottle (right figure).

or plot in Maple terminology – has been rendered, the viewer only provides control for color, line styles, perspective, axes, and polar coordinate rotation; sufficient control over the appearance of the visualization, but not much in the way of actually viewing it. Maple worksheets can be exported to a variety of document formats including html, rich text, and latex. The plots therein are exported via Graphics Interchange Format – although plots may be individually exported to various standard image formats such as eps, jpeg, and bmp. Certainly static images suffice for hardcopies, but with multimedia rich documents, such as mws and html, the dynamic qualities of visualizations should be preserved to convey the greatest amount of information.

Maples' programming environment is the Maple worksheet. With immediate access to the extensive mathematical function library, code is written, executed, and debugged inline. This makes coding small mathematical procedures relatively painless. However, on a larger scale such as package development, programming, compiling and debugging can quickly become cumbersome. While there are a few new developments that allow more control over the Maple interface, the absence of a software development kit (sdk), or application programmers' interface (api) makes it impossible to develop genuine, transparent plug-ins.

2.2 Interactive Visualization online with JavaView

JavaView is a sophisticated visualization appletcation and geometric software library. It offers a superior viewing environment that employs scaling, translation, quaternion rotation, customizable axes, colouring, materials and textures, transparency, anti-aliasing, depth cueing, animation, and camera control among other features. It is capable of importing and exporting 3d geometries in a wide variety of formats, and can perform modeling upon these geometries. It also offers an API, allowing custom plug-ins and visualizations to be developed for it. Furthermore, JavaView can be used as a standalone application or may be embedded as an applet in web pages for remote use via web browsers.

JavaView is a mature and portable geometry viewer. Numerous researchers and educators have utilized it for their own experiments and as a result have developed many general JavaView based visualization tools. However, a certain amount of Java programming and web development expertise is required to employ its full functionality. Unlike Maple, where an exhaustive function library and mathematics engine can be called upon to quickly input and execute mathematical programs, more programmatic care and custom code may be needed in JavaView to achieve similar results. The scope of the application is smaller, and therefore it may be easier to make use of other software applications to generate the visualization data or geometric models for the JavaView environment.

3 JavaViewLib - A New Maple Powertool

Maple makes little provision for application connectivity. However, it does allow for packages written in the Maple programming environment to be loaded into a Maple session. This development constraint combined with lack of control over the interface makes it impossible to create a plug-in in the true sense of the word. In the absence of an api, JVL enables the two applications to exchange data through flat files. Two file formats are utilized to allow geometric information to pass between the applications; one is an .mpl file that is nearly identical to the Maple plot data structure, the other, .jvx, is JavaView's native XML based file format for the storage of geometry models, animations and scenes. JVL parses and prepares Maple plot data in one of these two formats for import into JavaView, and as these files are the means of connectivity, JVL provides the mechanism by which the data in these files can be rendered in Maple. JVL also builds the necessary html code that browsers require to render these geometries in an embedded JavaView applet.

3.1 Maple Plot Data Structures

The maple plotting functions produce `PLOT` and `PLOT3D` data structures describing how Maple graphics are rendered in Maple's 2D or 3D viewer respectively. The two data structures produced are themselves internal Maple functions of the form `PLOT`(...) or `PLOT3D`(...). These structures can be viewed in a maple worksheet by assigning a plot function to a variable. Executing this variable on a Maple command line will invoke the functional data structure and render the image. Normally, the structures consist of the geometric data organized in labeled point lists followed by some stylistic information. For the some parts, JVL currently ignores stylistic information and uses JavaView's default options.

The `PLOT` data structure embeds four types of objects: points, curves, polygons, and text. In addition to these four types, the `PLOT3D` data structure embeds the mesh, grid and isosurface objects. A geometric point is represented by a list of floating point values; pi := [xi,yi] for the `PLOT` data structure and pi := [xi,yi,zi] for the `PLOT3D` data structure. Connected points are maintained in a geometric point list. A plot object is represented by a set of geometric points or geometric point lists wrapped in the corresponding label. For instance, the points object takes the form: `POINTS`(p1, p2,. . . ,pn), the curves object takes the form: `CURVES`([p1,p2,. . . ,pn], [q1,q2,. . . ,qn], . . . ,[r1,r2,. . . ,rn]), and the polygons object takes a form identical to the curves object, except the `POLYGONS` label indicates that the last point in each point list is to be connected to the first. The text object is nothing more than a label, a point and a string to display at that point: `TEXT`(p, “string”). The mesh, and grid objects are slightly more interesting.

Typically, plot functions involving parametric equations generate the mesh object. The mesh object maintains a matrix of geometric points based on a u-parameter resolution and a v-parameter resolution. This resolution is determined by specifying the amount by which to partition the u and v parameter ranges. The matrix, a list of geometric point lists, connects the geometric points sequentially by rows and by columns. For example, the following Maple plot function encapsulates the parametric representation for a sphere, and requires a discrete u,v-domain with which to compute the mapping. This is specified with a u-parameter range, a v-parameter range and the amount to partition by in the u and v directions with the `grid=[numU,numV]` argument. Storing the plot object reveals its data structure, and by examining the matrix of vertices embedded within the mesh object, we are able to determine the order in which the geometric points are connected.

```
[> s:=plots[sphereplot](1,u=0..2*Pi,v=0..Pi, grid=[4,5]);

s := PLOT3D(MESH(Array(1..4, 1..5, 1..3, [
[[0.,0.,1.],[.7,0.,.7],[1.,0.,0],[.7,0.,-.7],[0.,0.,-1.]],
[[0.,0.,1.],[-.35,.61,.7],[-.49,.86,0],[-.35,.61,-.7],[0.,0.,-1.]],
[[0.,0.,1.],[-.35,-.61,.7],[-.5,-.86,0.],[-.35,-.61,-.7],[0.,0.,-1.]],
[[0.,0.,1.],[.7,0.,.7],[1.,0.,0.],[.7,0.,-.7],[0.,0.,-1.]]])))
```

Most other plot functions in Maple generate the grid object. The grid object maintains a matrix of z-coordinate values based on an implicitly derived, discretely resolved Cartesian domain. The parameter ranges are also stored in the grid object, and are used with the row and column dimensions of the z-coordinate matrix to compute the corresponding x and y coordinate of the geometric point to be rendered. The geometric points are computed and connected sequentially by rows and by columns. For instance, the following Maple plot function specifies a Cartesian grid resolution with the grid=[numX,numY] argument and the x= x_1..x_2 and y=y_1..y_2 parameter ranges. The grid argument specifies the dimensions of the z-coordinate

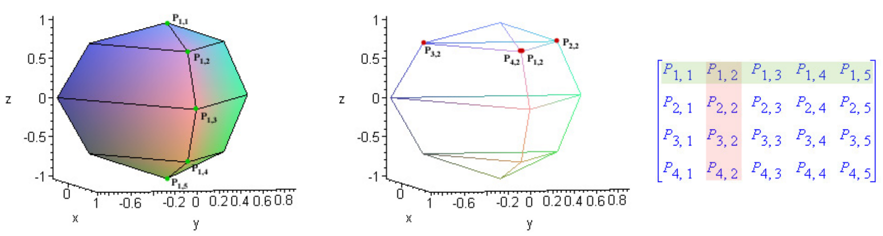


Fig. 3. Polyhedral meshes in the Maple Plot data structure.

matrix, whose values are incrementally computed with the given formula f. The first geometric point is resolved by P1,1 = [x1,y1,f(x1,y1)], and subsequent points are connected horizontally and vertically by Pi+1,j+1 [xi+(x2 - x1)/numX, yj+(y2 - y1)/numY, f(xi+(x2 - x1)/numX, yj+(y2 - y1)/numY)].

Finally, Maple achieves animation via the animate plot object. It contains a sequence of plot objects wrapped in a list where each list defines one frame in the animation. It takes the form: `ANIMATE`([Plot Object frame 1], [Plot Object frame 2], . . . , [Plot Object frame n]). For instance, the Maple command that follows animates a sequence of mesh objects. It uses the parameter t to describe how the geometry changes from frame to frame, and builds the set of mesh objects accordingly.

```
[> catHelAnim:=animate3d([cos(t)*cos(x)*cosh(y)+sin(t)*sin(x)*sinh(y),
   -cos(t)*sin(x)*cosh(y)+sin(t)*cos(x)*sinh(y),cos(t)*y+sin(u)*x],
   x=-Pi..Pi,y=-2..2,t=0..Pi,scaling=constrained);
catHelAnim = PLOT3D(ANIMATE([MESH(...)],[MESH(...)],...,[MESH(...)]),
      AXESLABELS(x,y,""),AXESSTYLE(FRAME),SCALING(CONSTRAINED))
[> runJavaView(catHelAnim);
```

3.2 Usage of JVL

The library functions available in JVL can be viewed after successfully loading the library into a maple session:

```
[> with(JavaViewLib);
```

```
[exportHTM, exportHTMLite, exportJVX, exportMPL, genTag, genTagLite,
 getInfo, import, runApplet, runAppletLite, runJavaView, runMarkupTree,
 set, viewGallery]
```

These functions employ JVL to set configuration parameters and build web pages, allow 3rd party geometries to be imported via JavaView, and enable maple plots to be exported in a variety of ways that interface with JavaView. Notice that some function names have the ‘Lite’ suffix. These functions make use of an alternative version of JavaView – optimized for size and speed – intended for use as a geometry viewer only. More details on the JVL commands are described in subsequent sections of this document.

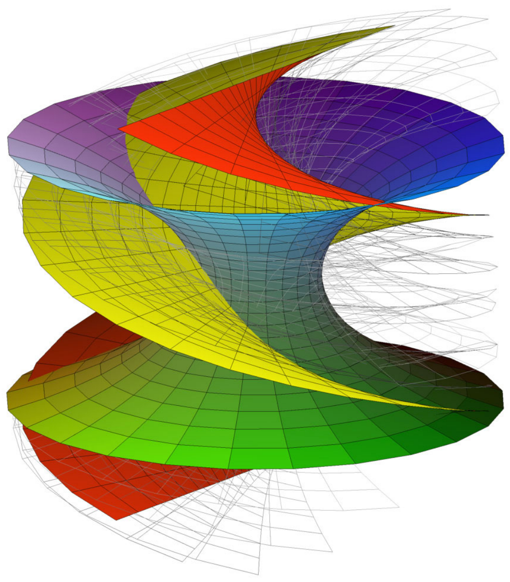


Fig. 4. Maple animations are shown as dynamic geometry in JavaView, optionally with a smooth morphing between keyframes. In this figure, the surface coloring of the helicoid and catenoid, and the display of grids of the in-betweenings was fine-tuned in JavaView.

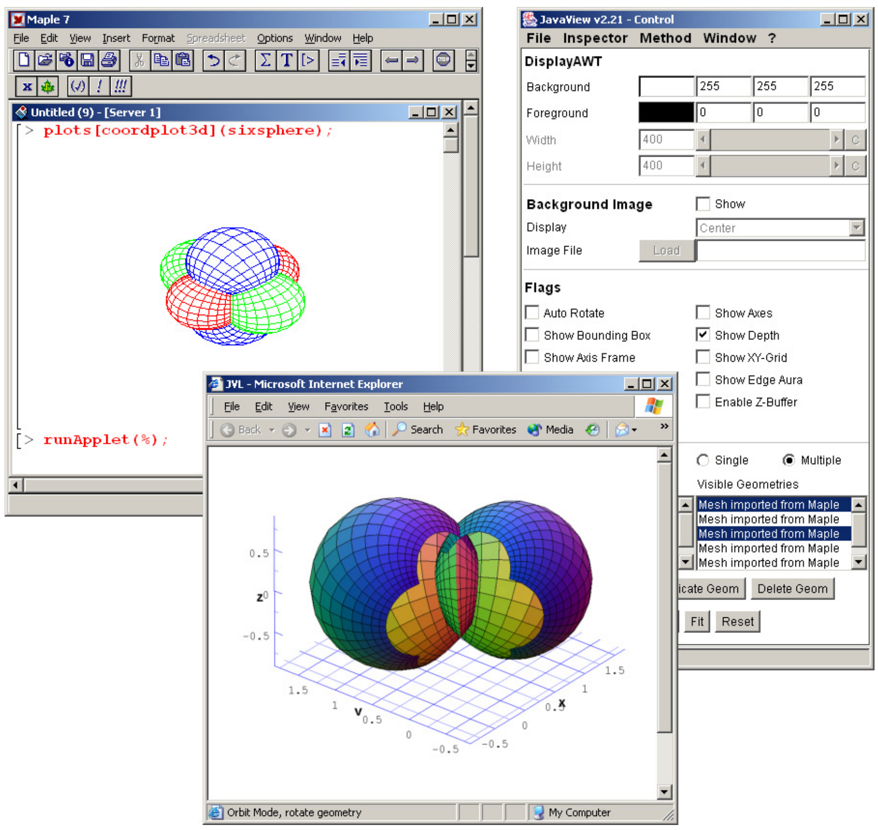
3.3 Basic Commands

The simplest way to use JVL is to wrap a 'run' function around a plot command. By default, a file called JVLExport.mpl will be created in the mpl folder of the installation directory. A call to `runJavaView` launches the JavaView standalone with the geometry contained in JVLExport.mpl. Once in JavaView, several operations can be applied to the geometry. For instance, when exporting a surface from Maple then adjacent polygons are not connected, that means, the common vertices appear multiple time. JavaView is able to identify these vertices and merge them to create a single seamless surface.

```
[> runJavaView(plot3d([4+x*cos(y),2*y,x*sin(y)],
   x=-Pi..Pi,y=-0..Pi,coords=cylindrical,grid=[2,40])):
```

The `runApplet` command is used to create an html file called JVLExport.htm containing the necessary applet tag and then to launch the defined browser to view it. Adding Maple plots to a JavaView enhanced web page allows geometries to be viewed remotely over the Internet. The following example also illustrates the additional flexibility of the JavaView system to analyse individual geometries of a complex scene, see Figure 5. Additional JVL commands will be discussed in the following sections.

```
[> runApplet(plots[coordplot3d](sixsphere));
```


Maple 7
File Edit View Insert Format Spreadsheet Options Window Help
Untitled (9) - [Server 1]
JavaView v2.21 - Control
File Inspector Method Window ?
JVL - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
0.5
z 0
-0.5
1.5
1
y 0.5
0
-0.5
-0.5
0
0.5 x
1
1.5
Orbit Mode, rotate geometry
My Computer

```
[> plots[coordplot3d](sixsphere);
[> runApplet(%);
```

DisplayAWT

Background	⬜	255	255	255
Foreground	⬛	0	0	0
Width	400			
Height	400			

Background Image: Show [ ]
Display: Center
Image File: Load

Flags

- [ ] Auto Rotate
- [ ] Show Axes
- [ ] Show Bounding Box
- [✓] Show Depth
- [ ] Show Axis Frame
- [ ] Show XY-Grid
- [ ] Show Edge Aura
- [ ] Enable Z-Buffer

Single [✓ Multiple]

Visible Geometries

Mesh imported from Maple
Mesh imported from Maple
Mesh imported from Maple
Mesh imported from Maple
Mesh imported from Maple

icate Geom | Delete Geom | Fit | Reset

Fig. 5. The JVL command `runApplet()` creates an interactive web page of any Maple plot.

3.4 Development of JVL

Maple's provision for application development comes in the way of packages. Packages are implemented with Maple's module construct, and are simply a collection of data and procedures, some of which are made public to the user. A number of Maple modules can be stored together in library, which is made up of three files: maple.lib maple.rep and maple.ind. The following partial code listing outlines JVL's basic module definition and the required calls to create the library. The module definition specifies the procedures to make public using the export identifier. Private procedures and data must be declared using the local identifier. Once these are initialized, they cannot be modified. Procedures and data declared with the global modifier are not publicly exposed, but are publicly accessible and may be modified. The option identifier, among other things, specifies the function to call when the package is loaded into a maple session. After the module is defined, the march (maple archive) command creates the library files in the specified directory. Once the library files are created the savelib command adds the Maple expression defined as 'JavaViewLib' to the archive.

```
[> JavaViewLib := module()
 option package, load = setup, `Copyright  Steven Dugaro 2001`:
 export import, exportHTM, runApplet, runJavaView, ..., set:
 global `type/JVLObject`:
 local  setup, import, ..., w1:

 # muted module constants; unmodifiable strings
  w1:="Unable to find the specified file.":
  ...
 # private function definitions
  setup := proc()
   interface(verboseproc=0):
   `type/JVLObject` := proc(x) ... end proc:
   setOS():
   buildIT():
  end proc;
  ...
 # public function definitions
  import := proc() ... end proc:
  ...

 end module:

[> march('create', '/JavaViewLib', 100);
[> savelib('JavaViewLib');
```

Using Maple's plot data structures and package mechanism it is straightforward to build application connectivity via parsers, file I/O and system calls. Two parsers were written – one within JavaView and one within JVL. The JVL parser extracts the geometric information from the plot data structures, and reproduces the information in JavaViews' jvx file format. JVL also outputs the plot structures into .mpl files from which the JavaView parser extracts the necessary geometric and stylistic information. These geometry files serve as the basic means of information exchange between JavaView and Maple. The jvx file format is intended for the export of stylistic free geometries in a human readable file format. The .mpl file format should be used to export stylistic geometries in a compact file format. These files are passed to JavaView via a command line argument when it is invoked as a standalone, or via an applet tag parameter when it is rendered in a browser. JVL also allows plot data to be included as an applet parameter in the applet tag so that all the necessary information may be contained in a single html file. Using simple file I/O, JVL creates the geometry files, the html files or both then employs system calls to launch JavaView or the user's browser from within Maple. The following code snippet shows a brief example of a JVL library procedure that performs plot parsing and JVX file generation. Note, saving Maple plots as JVX file will allow JVL to include additional rendering commands like setting transparency of a geometry. However, exporting Maple plots to JavaView via MPL files will work fine, too.

```
[> exportJVX := proc()
# PUBLIC: save a Maple plot in JavaView's JVX file format

 if nargs = 1 then
  if type(args[1], JVLObject) then
   oargs:= getIOstring("JVLExport",jvx),args[1]:fi:
 else
  oargs:=check2args(args):
  oargs:=getIOstring(oargs[1],"jvx"),oargs[2]:fi:

 try
  fd:= fopen(getIOstring(oargs[1],jvx), WRITE):
  fprintf(fd,"%s", JVXHEADER):
  fprintf(fd," <jvx-model>\n\t<geometries>\n"):
  for l from 1 to nops(oargs[2]) do
   a:=op(l,oargs[2]):
   if op(0,a) = MESH then
   ...
   elif op(0,a) = GRID then
   ...
    elif op(0,a) = POLYGONS then
   ...
   elif op(0,a) = CURVES then
    ...
   elif op(0,a) = POINTS then
    ...
   else # handle other Maple objects including styles.
    ...
   fi:
  od:
  fprintf(fd,"\n\t</geometries>\n</jvx-model>"):
  fclose(fd):
  return oargs[1]:
 catch:
  error("Could not write to file ", oargs[1], lastexception):
 end try:
end proc:
```

4 Importing and Exporting Geometries

JVL has extended the capabilities of Maple to make geometric information highly portable. For the first time it is possible to export Maple plots to a variety of formats and import geometries from a variety of formats into Maple. One is finally able to export Maple worksheets into an html file where the dynamic qualities of the plot is preserved.

As it is typically encouraged to keep similar file types grouped together when developing web pages, JVL maintains a working directory to organize its exports. Its working directory contains four subfolders: 1) an ./mpl subfolder for mpl files, 2) a ./jvx subfolder for jvx files 3) an ./htm subfolder for html files, and 4) a ./jars subfolder for the JavaView applet. The working directory defaults to the JVL installation directory, so it is recommended that a working directory be set before exporting.

4.1 Import of Geometry Files into Maple

JVL does not implement its own file format parsers; these parsers are already implemented in JavaView. These parsers cannot be accessed from within JVL since application connectivity is weak. As a result, importing 3rd party geometries into Maple requires an intermediate step. After the geometry has been loaded into JavaView, it must be saved down as an mpl file. At this point JVL's `import` command can be called to pull the geometry into Maple. The following example and Figure 6 illustrate this.

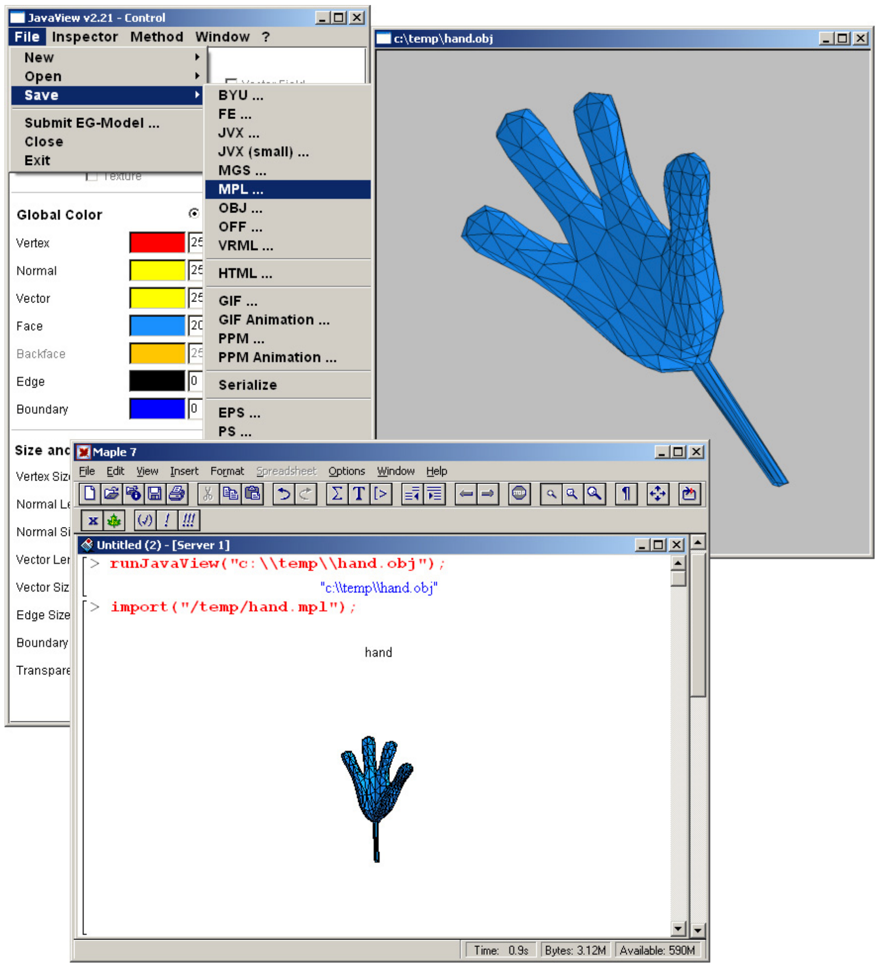
```
[> runJavaView("/temp/hand.obj");
[> import("/temp/hand.mpl");
```

4.2 Exporting Maple Plots to Web Pages

The three fundamental JVL export functions are `exportHTM`, `exportMPL`, and `exportJVX`. They provide three different contexts in which to export Maple plot data. The former is used to generate html pages that either link to or contain the plot information and embed the JavaView applet. The latter two, are for generating the respective geometric files only. These functions require 1 argument – the plot object. A second optional argument may be given to specify the filename and path to which the file is to be exported. Maples plots commands can be wrapped in the export functions or defined in a variable to be passed by reference. Here we define a simple cube in the box variable:

```
[> box:=plots[polyhedraplot]([0,0,0],polytype=hexahedron):
```

The `exportMPL` interface is used to export a stylistic, compact representation of the Maple plot. JavaView interprets most of the plot attributes, and selectively discards others. The following command exports the geometry into a file called mplBox.mpl into the mpl folder of the current working directory.



```
> runJavaView("c:\\temp\\hand.obj");
                    "c:\\temp\\hand.obj"
> import("/temp/hand.mpl");
                    hand
```

Fig. 6. Import a wealth of 3D file formats into Maple via JavaView's geometry loaders. For example, OBJ is the standard file format of Java3D and Wavefront, and accepted by rendering software like Maya.

```
[> exportMPL(box, mplBox):
```

The `exportJVX` interface is used to export a minimal, legible representation of the Maple plot. However, the display can be embellished with the addition of attributes and other jvx tags to the jvx file. The following command exports the geometry to a file called jvxBox.jvx in the temp folder of the root directory.

```
[> exportJVX("/temp/jvxBox", box):
```

The `exportHTM` interface is used to generate and couple html pages with exported Maple plots. This can be done in one of three ways: 1) embed the data within the html page itself, 2) generate and link to a geometry file, or 3) create the html page for an existing geometry file. Appending a filename extension qualifies the method of plot export. Embedding the data within the html file is the default method for this function and so no extension is needed, however appending '.jvx' or '.mpl' will export the plot to a separate file and creates the html page that links to it. JVL normally exports with respect to the working directory so that all files can be relatively referenced, nevertheless, exporting to an arbitrary path will copy all the relevant files to that directory. The following commands demonstrate these four possible usage scenarios.

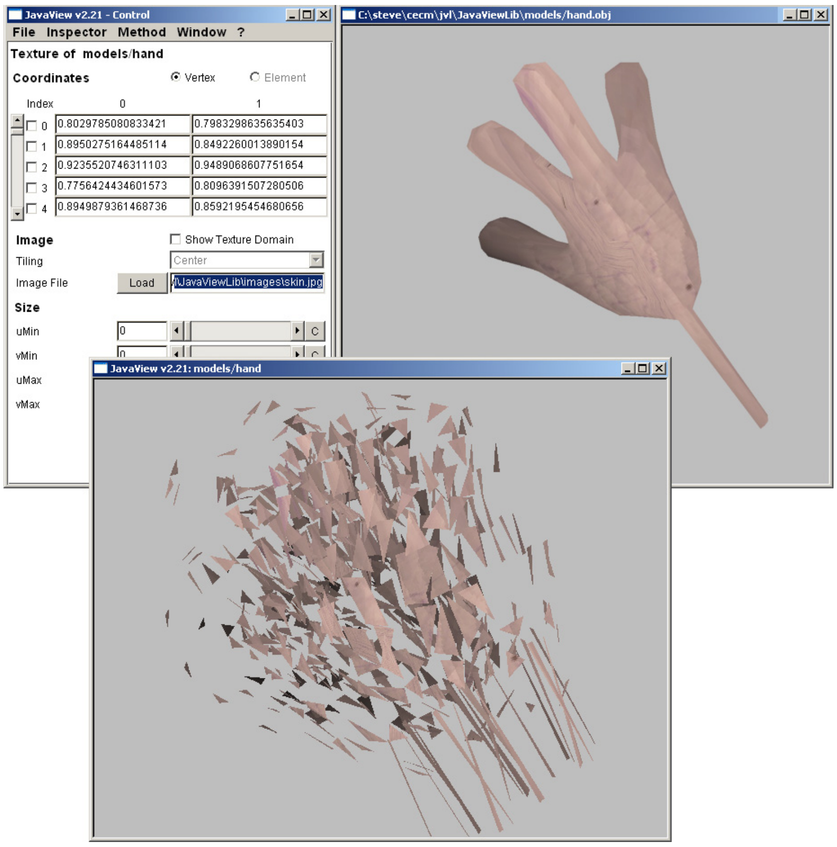
```
[> exportHTM(box,"box");        # export the plot into an html file
[> exportHTM(box,"box.mpl");    # generate box.mpl, box.htm & link
[> exportHTM(box,"jvxBox.jvx"); # generate jvxBox.jvx and jvxBox.htm
[> exportHTM(box,"/temp/");     # copy jars, JVLExport.htm to /temp
```

4.3 More on the run* Commands

The JVL `run*` commands are basically wrappers around the above export functions. They do little more than specify the export method and launch the appropriate application. Consequently, the argument guidelines are the same as the export functions. A filename and/or path is optional, defaulting to JVLExport in the current working directory, a qualifying file extension specifies the method of export, and arbitrary paths copy the necessary files to the specified location. However, unlike `exportHTM`, existing html files are opened for viewing and not overwritten.
The JavaView standalone application is interfaced with Maple using the runJavaView command. It contains the most complete compilation of the JavaView modules as web considerations need not be taken into account. This interface is provided for the use of JavaView on your local machine, and automatically launches JavaView from within Maple. Once a model has been loaded into JavaView, Javaview's geometry tools may be utilized. This includes materials features such as texture mapping, modeling features such as triangulation, and effects features such as explosions. The following function calls illustrate how to typically make use of this interface function, see Figure 7.

```
[> runJavaView():    # launch the JavaView application
[> runJavaview(box): # launch JavaView with a Maple plot
[> runJavaview(box,myBox.mpl): # launch and save a Maple plot
[> runJavaView("models/hand.obj"): # load a 3rd party geometry
```

As an applet, JavaView can be used interactively over the Internet. The `runApplet` function is able to expedite this process by exporting maple plots


JavaView v2.21 - Control
File Inspector Method Window ?
C:\steve\cecm\jvl\JavaViewLib\models/hand.obj
JavaView v2.21: models/hand
Load

Texture of models/hand

Coordinates [✓ Vertex] Element

Index	0	1
0	0.8029785080833421	0.7983298635635403
1	0.8950275164485114	0.8492260013890154
2	0.9235520746311103	0.9489068607751654
3	0.7756424434601573	0.8096391507280506
4	0.8949879361468736	0.8592195454680656

Image ☐ Show Texture Domain

Tiling: Center

Image File: \JavaViewLib\images\skin.jpg

Size

uMin: 0

vMin: 0

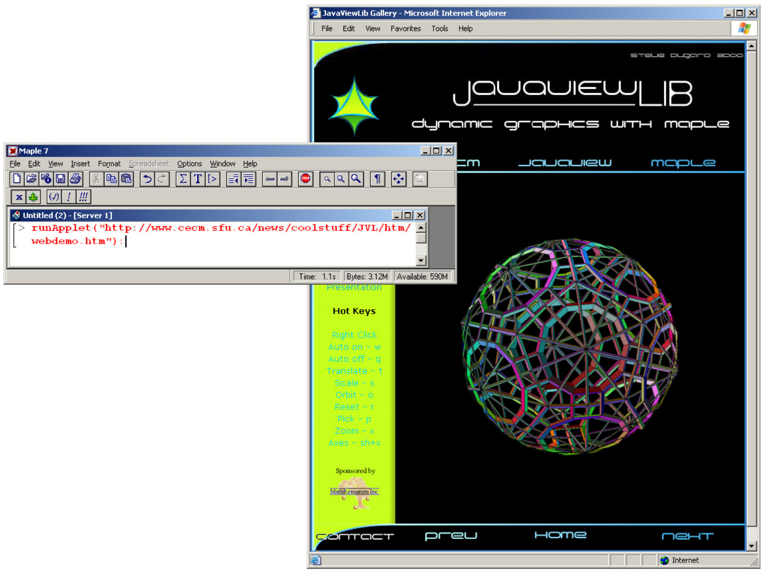
uMax

vMax

Fig. 7. JavaView's advanced modeling tools allow for the fine tuning of geometric shapes.

to a 'skeletal web page', which can then be fleshed out into a final html document. This 'skeletal web page' simply contains the applet tag that embeds the JavaView applet. Tags in general are directory structure dependent as they point to the files with which the browser should render the page. Therefore it is important to keep the structure maintained by JVL in your working path - relocating files would require you to manually adjust the tag's definition. After building and exporting the necessary files, `runApplet` will automatically launch the defined browser from within Maple for viewing. The following examples demonstrate some typical uses for this function.

```
[> url:= "http://www.cecm.sfu.ca/news/coolstuff/JVL/htm/webdemo.htm":
[> runApplet(url): # open a web page from within Maple
[> runApplet(box): # launch a browser with a Maple plot
[> runApplet(jvxBox.jvx): # launch and build page for existing file
[> runApplet(box,myBox.mpl): # launch, build page, and save a plot
[> runApplet("/models/hand.obj"): # launch a 3rd party geometry
[> runApplet(box,"/temp/box.htm"): # launch, build, copy to path
```



```
[> runApplet("http://www.cecm.sfu.ca/news/coolstuff/JVL/htm/
   webdemo.htm"):
```

Maple 7
File Edit View Insert Format Spreadsheet Options Window Help
Untitled (2) - [Server 1]
Time: 1.1s Bytes: 3.12M Available: 590M
JavaViewLib Gallery - Microsoft Internet Explorer
File Edit View Favorites Tools Help
JavaViewLib
dynamic graphics with maple
javaview
maple
Presentation
Hot Keys
Right Click
Auto on - w
Auto off - q
Translate - t
Scale - s
Orbit - o
Reset - r
Pick - p
Zoom - x
Axes - sh+x
Sponsored by
contact
prev
home
next
Internet

Fig. 8. The original homepage of JavaViewLib launched from inside Maple.

5 Additional Features

5.1 JVL State Information

JVL maintains a small amount of state information to assist in the configuration of the JavaView viewer. For the most part, these states specify how the JavaView applet is to be rendered in a browser. These states can be set to specify the size of the JavaView viewport, its background image, the current working path as well as some viewing initializations such as autorotation, axes, and depth cueing. The list of state information can be obtained with the following function:

```
[> getInfo();

JavaViewLib State Information
------------------------+-------
[W ] Applet Width       | 400
[H ] Applet Height      | 400
[A ] Applet Alignment   | Center
```

```
[R ] AutoRotate         | 1. 1. 1.
[X ] Axes               | Hide
[BC] Background Colour  | 255 255 255
[BI] Background Image   | images/jvl.jpg
[B ] Border             | Hide
[BB] Bounding Box       | Show
[BR] Browser            | iexplore
[V ] Camera Direction   | 1. 2. 3.
[DC] Depth Cueing       | Hide
[EA] Edge Aura          | Show
[WK] Working Path       | C:\Program Files\Maple 6\JavaViewLib\
     Installation Path  | C:\Program Files\Maple 6\JavaViewLib\
     Operating System   | Windows NT/2000
```

JVL states can be configured with the `set` command by specifying a list of attribute = value pairs. Most binary states can be toggled by either assigning a `show/hide` or `on/off` value, or by simply including its handle in the list. Here we set the working path for the export project and specify that all subsequent tags are to be rendered with a left aligned 200 pixel by 200 pixel viewport, in auto-rotate model with axes, and the currently defined background image.

```
[> set(wp="c:\\temp\\myGeoms\\", width=200, height=200, axes,
       autorotate, bg=image, align=left):
[> runApplet(plots[polyhedraplot]([0,0,0], polytype=hexahedron));
```

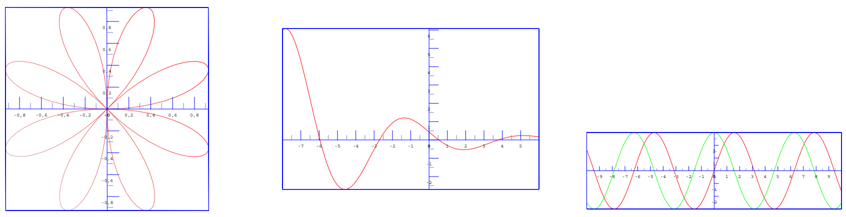
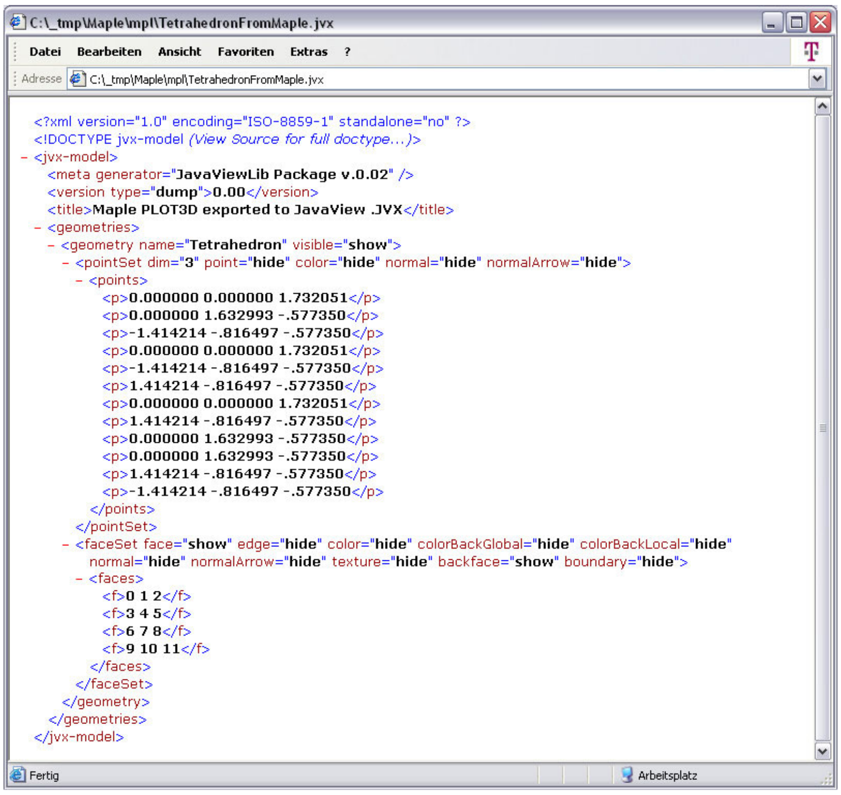


Fig. 9. 2D graphs exported to a JavaView applet will keep axis and projection settings.

5.2 Markup Tree of the JVX Geometry

The `runMarkupTree` command simply exports a plot in the jvx format, and launches a browser to view it. The markup tree is an XML representation of the geometric data contained in a plot, and consists of tags that represent points, faces, and geometries. The listing in Figure 10 is the markup tree for a tetrahedron. Notice that the Maple plot, and therefore the .mpl


C:_tmp\Maple\mpl\TetrahedronFromMaple.jvx
Datei Bearbeiten Ansicht Favoriten Extras ?
Adresse C:_tmp\Maple\mpl\TetrahedronFromMaple.jvx
Fertig
Arbeitsplatz

```
  <?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
  <!DOCTYPE jvx-model (View Source for full doctype...)>
- <jvx-model>
    <meta generator="JavaViewLib Package v.0.02" />
    <version type="dump">0.00</version>
    <title>Maple PLOT3D exported to JavaView .JVX</title>
  - <geometries>
    - <geometry name="Tetrahedron" visible="show">
      - <pointSet dim="3" point="hide" color="hide" normal="hide" normalArrow="hide">
        - <points>
            <p>0.000000 0.000000 1.732051</p>
            <p>0.000000 1.632993 -.577350</p>
            <p>-1.414214 -.816497 -.577350</p>
            <p>0.000000 0.000000 1.732051</p>
            <p>-1.414214 -.816497 -.577350</p>
            <p>1.414214 -.816497 -.577350</p>
            <p>0.000000 0.000000 1.732051</p>
            <p>1.414214 -.816497 -.577350</p>
            <p>0.000000 1.632993 -.577350</p>
            <p>0.000000 1.632993 -.577350</p>
            <p>1.414214 -.816497 -.577350</p>
            <p>-1.414214 -.816497 -.577350</p>
          </points>
        </pointSet>
      - <faceSet face="show" edge="hide" color="hide" colorBackGlobal="hide" colorBackLocal="hide"
          normal="hide" normalArrow="hide" texture="hide" backface="show" boundary="hide">
        - <faces>
            <f>0 1 2</f>
            <f>3 4 5</f>
            <f>6 7 8</f>
            <f>9 10 11</f>
          </faces>
        </faceSet>
      </geometry>
    </geometries>
  </jvx-model>
```

Fig. 10. View a Maple plot as formatted XML document, and collaps and expand individual parts of the markup tree. The JVL command `runMarkupTree()` makes use of JavaView's JVX file format and the formatting capabilities of the Internet Explorer. Here the vertices and faces of a tetrahedron are listed.

format, contains many redundant points. These points can be merged with JavaView's 'Identify Points' modeling command and preserved by saving in the .jvx format.

```
[> runMarkupTree(plots[polyhedraplot]([0,0,0], polytype=tetrahedron));
```

5.3 Create and Configure an Applet Tag

On occasion, it may be quicker to simply generate the applet tags for a series of plots, instead of exporting an html page for each of them. The genTag function returns the applet tag as it would be rendered in the html document if exported otherwise. These tags can then be cut and paste into a single html document kept relative to the working directory – i.e. in the htm subfolder. The following example quickly generates 3 orthogonal views and

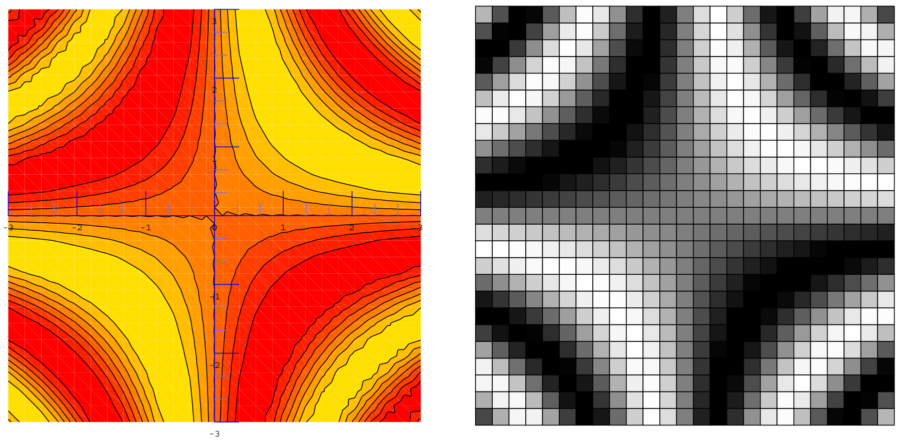


Fig. 11. JavaView recognizes a large variety of different Maple plots including contour and density plots.

one perspective view with auto-rotation using the lite version of the JavaView applet.

```
[> b := plot3d((1.3)^x * sin(y),x=-1..2*Pi,y=0..Pi,
               coords=spherical,style=patch):
[> set(reset):
[> set(viewDir="0 -1 0", bg="200 200 200", border="on", axes="off",
       width=300, height=300):
[> exportHTM(b,bounty.mpl):
[> set(viewDir="0 1 0"):
[> genTagLite(bounty.mpl);
 <APPLET CODE='jvLite.class' CODEBASE='../' ARCHIVE='jars/jv_lite.zip'
  WIDTH='300' HEIGHT='300' ID='JVLAPPLET' ALT='JVL - MAPLE Export'>
  <PARAM NAME='Background' VALUE='200 200 200'>
  <PARAM NAME='Border' VALUE='Show'>
  <PARAM NAME='ViewDir' VALUE='0 1 0'>
  <PARAM NAME='Model' VALUE='mpl/bounty.mpl'>
 </APPLET>

[> set(viewDir="0 0 1"):
[> genTagLite(bounty.mpl);
 <APPLET CODE='jvLite.class' CODEBASE='../' ARCHIVE='jars/jv_lite.zip'
  WIDTH='300' HEIGHT='300' ID='JVLAPPLET' ALT='JVL - MAPLE Export'>
  <PARAM NAME='Background' VALUE='200 200 200'>
  <PARAM NAME='Border' VALUE='Show'>
  <PARAM NAME='ViewDir' VALUE='0 0 1'>
  <PARAM NAME='Model' VALUE='mpl/bounty.mpl'>
 </APPLET>

[> set(viewDir="1 1 1", axes="boundingbox", rotate):
[> genTagLite(b,bounty.jvx);
```

```
<APPLET CODE='jvLite.class' CODEBASE='../' ARCHIVE='jars/jv_lite.zip'
 WIDTH='300' HEIGHT='300' ID='JVLAPPLET' ALT='JVL - MAPLE Export'>
 <PARAM NAME='Axes' VALUE='Show'>
 <PARAM NAME='BoundingBox' VALUE='Show'>
 <PARAM NAME='Background' VALUE='200 200 200'>
 <PARAM NAME='Border' VALUE='Show'>
 <PARAM NAME='AutoRotate' VALUE='Show'>
 <PARAM NAME='ViewDir' VALUE='1 1 1'>
 <PARAM NAME='Model' VALUE='jvx/bounty.jvx'>
</APPLET>
```

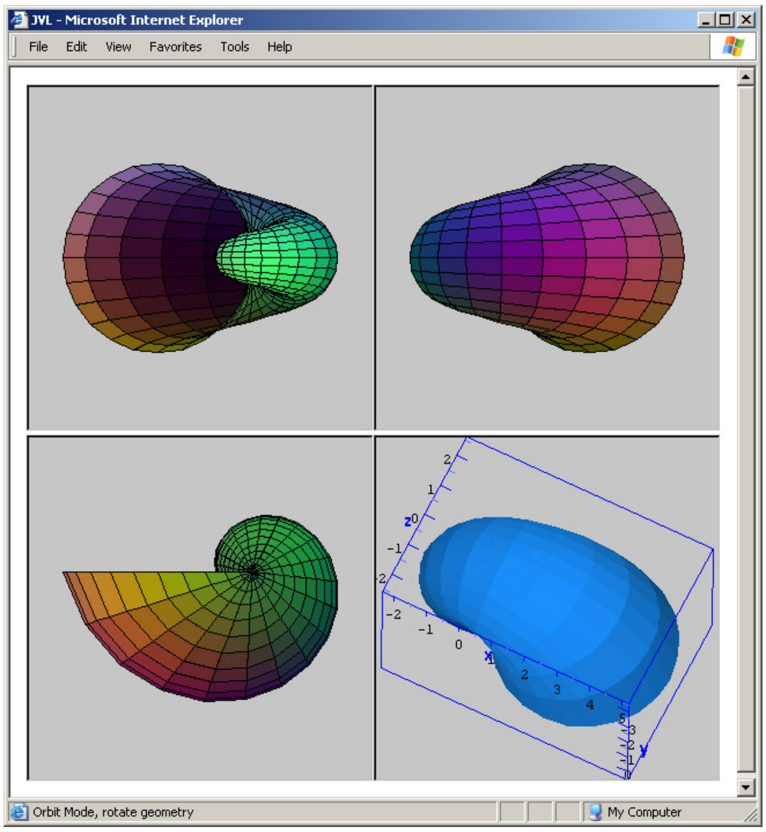

JVL - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Orbit Mode, rotate geometry
My Computer
z
x
y

Fig. 12. The JVL commands `genTag()` create an applet tag of a Maple plot. Here `genTag()` was used to show different projections of the same geometry.

5.4 Creating a Web Gallery of Maple Plots

The quickest way to make exported geometries web ready is to let JVL do it. The `viewGallery` command builds a frame based geometry gallery with a table of contents that links to all exports in the htm folder of the current working directory. The following commands export a 2D geometry to the htm folder, and build the necessary html files for the gallery at the top level of the current working directory. This provides a quick way to publish visualization projects on the Internet.

```
[> exportHTM(plot([sin(4*x),x,x=0..2*Pi],coords=polar):
[> viewGallery();
```

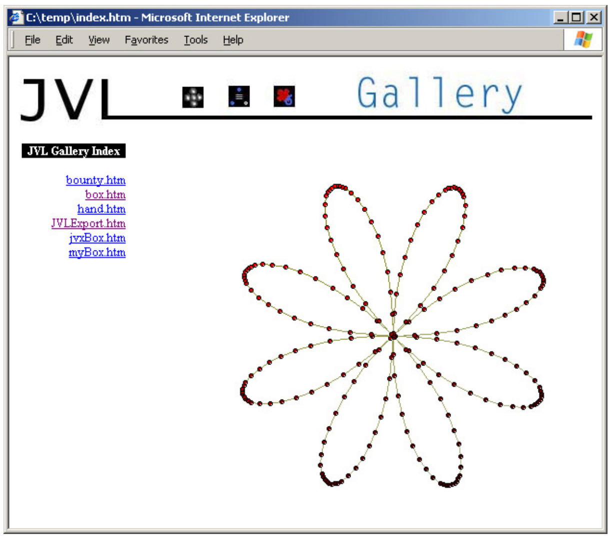


Fig. 13. Automatically create a web gallery of all Maple plots in your repository with the JVL command `viewGallery()`.

6 Conclusion and Outlook

By establishing even a basic level of application connectivity, the functionality of both JavaView and Maple was enhanced. Maple was afforded greater visualization and web functionality while JavaView's geometry generation capabilities were extended to match that of Maple. The JavaViewLib has made its mark in the sand as a proponent of application connectivity. By bridging these applications, the JavaViewLib broadens the toolset available for the research and teaching of mathematics.

At the time of its development, the JavaViewLib made the best possible use of the connectivity resources made available by Maple and JavaView. However, in the year since its release, there is now room for improvement; both Maple and JavaView appear to be making considerable advancements in the provision for third party control. JavaView has introduced a secondary XML file format for the initialization and preservation of display and camera properties. Through this format, known as jvd, precise control over camera, lighting, and viewport properties can be specified and launched with JavaView. This is a step in the right direction, but covers only a small portion of JavaView's rich feature set. Ideally, this file format will mature into a full featured scripting language that allows the broad range of JavaView operations to be applied to the geometry or geometries loaded into the viewer. Two new technologies introduced with Maple 8, known as Maplets and MapleNet appear to be moving in the direction of third party integration. However, the two fall just short of providing a transparent look and feel for third party plugins.

7 Downloading JavaViewLib

The JavaViewLib has become an official Maple Powertool, and may be obtained from the MapleSoft website [3] at

`http://www.mapleapps.com/powertools/javalib/javalib.shtml`.

The original website of JVL [1][2] as well as new releases and updates reside at the CECM website under

`http://www.cecm.sfu.ca/news/coolstuff/JVL/htm/webdemo.htm`.

The JavaView [5] visualization environment, which also includes the parser for Maple plots, is contained in the JVL download but may be upgraded independently by replacing the JavaView directory with newer versions from the JavaView homepage

`http://www.javaview.de`.

Package downloads include a tutorial Maple worksheet *gettingStarted.mws* and a *readme.txt* file for installation instructions.

References

1. S. Dugaro. JavaViewLib homepage. `www.cecm.sfu.ca/news/coolstuff/JVL/htm/webdemo.htm`.
2. S. Dugaro. JavaViewLib - a visualization powertool. In *Proc. of the Maple Summer Workshop*. Waterloo Maple Inc., 2002.
3. Maple Waterloo. Homepage. `http://www.maplesoft.com`.
4. Maple Waterloo. Powertools homepage. `http://www.mapleapps.com/powertools/`.
5. K. Polthier. JavaView homepage, 1998–2002. `http://www.javaview.de/`.
6. K. Polthier, S. Khadem-Al-Charieh, E. Preuß, and U. Reitebuch. Publication of interactive visualizations with JavaView. In J. Borwein, M. H. Morales, K. Polthier, and J. F. Rodrigues, editors, *Multimedia Tools for Communicating Mathematics*. Springer Verlag, 2002. `http://www.javaview.de`.
7. Wolfram Research. Homepage. `http://www.wolfram.com`.