# Mayavi: Making 3D Data Visualization Reusable

Prabhu Ramachandran (prabhu@aero.iitb.ac.in) – *Indian Institute of Technology Bombay, Powai, Mumbai 400076* India
Gaël Varoquaux (gael.varoquaux@normalesup.org) – *NeuroSpin, CEA Saclay, Bât 145, 91191 Gif-sur-Yvette* France

**Mayavi is a general-purpose 3D scientific visualization package. We believe 3D data visualization is a difficult task and different users can benefit from an easy-to-use tool for this purpose. In this article, we focus on how Mayavi addresses the needs of different users with a common code-base, rather than describing the data visualization functionalities of Mayavi, or the visualization model exposed to the user.**

Mayavi2 is the next generation of the Mayavi-1.x package which was first released in 2001. Data visualization in 3D is a difficult task; as a scientific data visualization package, Mayavi tries to address several challenges. The Visualization Toolkit [VTK] is by far the best visualization library available and we believe that the rendering and visualization algorithms developed by VTK provide the right tools for data visualization. Mayavi therefore uses VTK for its graphics. Unfortunately, VTK is not entirely easy to understand and many people are not interested in learning it since it has a steep learning curve. Mayavi strives to provide interfaces to VTK that make it easier to use, both by relying on standard numerical objects (numpy arrays) and by using the features of Python, a dynamical language, to offer simple APIs.
There are several user requirements that Mayavi strives to satisfy:

- A standalone application for visualization,
- Interactive 3D plots from IPython like those provided by pylab,
- A clean scripting layer,
- Graphical interfaces and dialogs with a focus on usability,
- Visualization engine for embedding in user dialogs box,
- An extensible application via an application framework like Envisage,
- Easy customization of the library and application,

The goal of Mayavi is to provide flexible components to satisfy *all* of these needs. We feel that there is value in reusing the core code, not only for the developers, from a software engineering point of view, but also for the users, as they can get to understand better the underlying model and concepts using the different facets of Mayavi.

Mayavi has developed in very significant ways over the last year. Specifically, every one of the above requirements have been satisfied. We first present a brief overview of the major new functionality added over the last year. The second part of the paper illustrates how we achieved the amount of reuse we have with Mayavi and what we have learned in the process of implementing this. We believe that the general ideas involved in making Mayavi reusable in these different contexts are applicable to other projects as well.

## Mayavi feature overview

Starting with the Mayavi 3.0.0 release[1], there have been several significant enhancements which open up different ways of using Mayavi. We discuss each of these with examples in the following.

### The `mayavi2` application

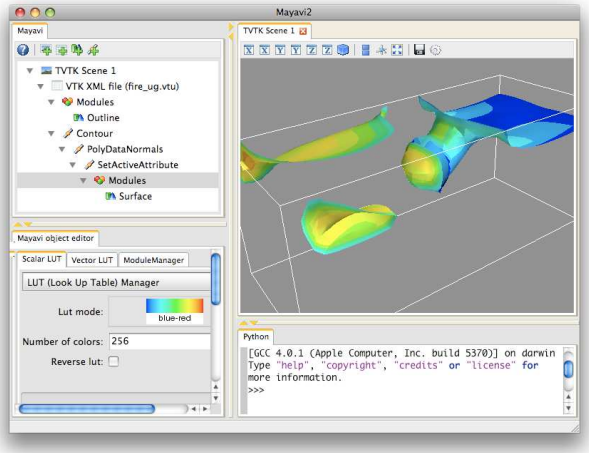
`mayavi2` is a standalone application that provides an interactive user interface to load data from files (or other sources) and visualize them interactively. It features the following:

- A powerful command line interface that lets a user build a visualization pipeline right from the command line,
- An embedded Python shell that can be used to script the application,
- The ability to drag and drop objects from the mayavi tree view on to the interpreter and script the dropped objects,
- Execution of arbitrary Python scripts in order to rapidly script the application,
- Full customization at a user level and global level. As a result, the application can be easily tailored for specific data files or workflows. For instance, the Imperial College's Applied Modeling and Computation Group has been extending Mayavi2 for triangular-mesh-specific visualizations.

[1] The name "Mayavi2" refers to the fact that the current codebase is a complete rewrite of the first implementation of Mayavi. We use it to oppose the two very different codebases and models. However the revision number of the Mayavi project is not fixed to two. The current release number is 3.0.1, although the changes between 2 and 3 are evolutionary rather than revolutionary.

- Integration into the Envisage application framework. Users may load any other plugins of their choice to extend the application. Envisage is a plugin-based application framework, similar to Eclipse, for assembling large applications from loosely-coupled components. The wing-design group at Airbus, in Bristol, designs wing meshes for simulations with a large application built with Envisage using Mayavi for the visualization.

Shown below is a visualization made on the mayavi user interface.




*Screenshot of the Mayavi application.*

### The `mlab` interface

Mayavi's `mlab` interface provides an easy scripting interface to visualize data. It can be used in scripts, or interactively from an IPython session in a manner similar to matplotlib's pylab interface. `mlab` features the following:

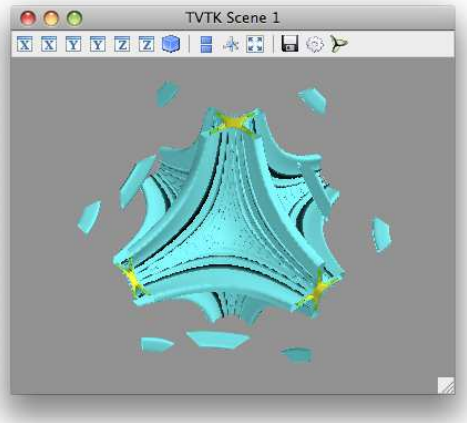
- As easy to use as possible.
- Works in the `mayavi2` application also.
- Trivial to visualize numpy arrays.
- Full power of mayavi from scripts and UI.
- Allows easy animation of data without having to recreate the visualization.

A simple example of a visualization with mlab is shown below:

```
from enthought.mayavi import mlab
from numpy import ogrid, sin

x, y, z = ogrid[-10:10:100j,
                -10:10:100j,
                -10:10:100j]

ctr = mlab.contour3d(sin(x*y*z)/(x*y*z))
mlab.show()
```




*Visualization created by the above code example.*

`mlab` also allows users to change the data easily. In the above example, if the scalars needs to be changed it may be easily done as follows:

```
new_scalars = x*x + y*y*0.5 + z*z*3.0
ctr.mlab_source.scalars = new_scalars
```

In the above, we use the `mlab_source` attribute to change the scalars used in the visualization. After setting the new scalars the visualization is immediately updated. This allows for powerful and simple animations.

The core features of `mlab` are all well-documented in a full reference chapter of the user-guide [M2], with examples and images.

`mlab` also exposes the lower-level mayavi API in convenient functions via the `mlab.pipeline` module. For example one could open a data file and visualize it using the following code:

```
from enthought.mayavi import mlab
src = mlab.pipeline.open('test.vtk')
o = mlab.pipeline.outline(src)
cut = mlab.pipeline.scalar_cut_plane(src)
iso = mlab.pipeline.iso_surface(src)
mlab.show()
```

`mlab` thus allows users to very rapidly script Mayavi.

### Object-oriented interface

Mayavi features a simple-to-use, object-oriented interface for data visualization. The `mlab` API is built atop this interface. The central object in Mayavi visualizations is the `Engine`, which connects the different elements of the rendering pipeline. The first `mlab` example can be re-written using the `Engine` object directly as follows:

```
from numpy import ogrid, sin
from enthought.mayavi.core.engine import Engine
from enthought.mayavi.sources.api import ArraySource
from enthought.mayavi.modules.api import IsoSurface
from enthought.pyface.api import GUI

e = Engine()
e.start()
scene = e.new_scene()

x, y, z = ogrid[-10:10:100j,
                -10:10:100j,
                -10:10:100j]
data = sin(x*y*z)/(x*y*z)

src = ArraySource(scalar_data=data)
e.add_source(src)
e.add_module(IsoSurface())
GUI().start_event_loop()
```

Clearly `mlab` is a lot simpler to use. However, the raw object-oriented API of mayavi is useful in its own right, for example when using mayavi in an object-oriented context where one may desire much more explicit control of the objects and their states.

## Embedding a 3D visualization

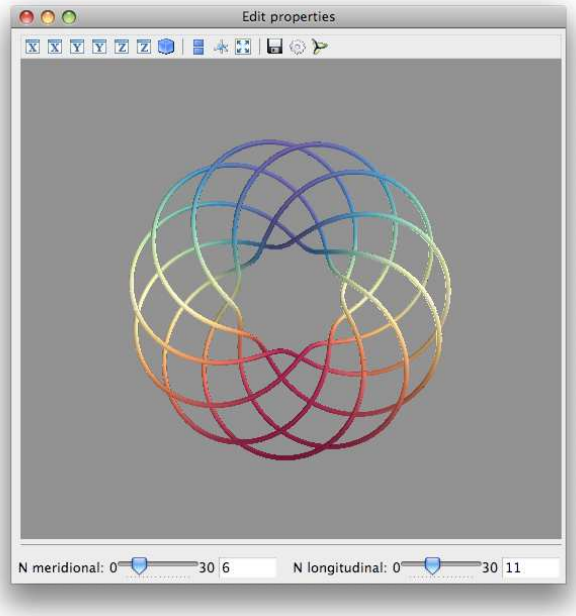
One of the most powerful features of Mayavi is the ability to embed it in a user interface dialog. One may do this either with native Traits user interfaces or in a native toolkit interface.

### Embedding in TraitsUI

The TraitsUI module, used heavily throughout Mayavi to build dialogs, provides user-interfaces built on top of objects, exposing their attributes. The graphical user-interface is created in a fully descriptive way by associating object attributes with graphical editors, corresponding to views in the MVC pattern. The objects inheriting from the `HasTraits` class, the workhorse of Traits, have an embedded observer pattern, and modifying their attributes can fire callbacks, allowing the object to be manipulated live, e.g.through a GUI.

TraitsUI is used by many other projects to build graphical, interactive applications. Mayavi can easily be embedded in a TraitsUI application to be used as a visualization engine.

Mayavi provides an object, the `MlabSceneModel`, that exposes the `mlab` interface as an attribute. This object can be viewed with a `SceneEditor` in a TraitsUI dialog. This lets one use Mayavi to create dynamic visualizations in dialogs. Since we are using Traits, the core logic of the dialog is implemented in the underlying object. The `modifying_mlab_source.py` example can be found in the Mayavi examples and shows a 3D line plot parametrized by two integers. Let us go over the key elements of this example, the reader should refer to the full example for more details. The resulting UI offers slider bars to change the values of the integers, and the visualization is refreshed by the callbacks.




*A Mayavi visualization embedded in a custom dialog.*

The outline of the code in this example is:

```
from enthought.tvtk.pyface.scene_editor import \
                SceneEditor
from enthought.mayavi.tools.mlab_scene_model \
                import MlabSceneModel
from enthought.mayavi.core.pipeline_base \
                import PipelineBase

class MyModel(HasTraits):
    [...]

    scene = Instance(MlabSceneModel, ())
    plot = Instance(PipelineBase)

    # The view for this object.
    view = View(Item('scene',
                     editor=SceneEditor(),
                     height=500, width=500,
                     show_label=False),
                [...])

    def _plot_default(self):
        x, y, z, t = curve(self.n_merid, self.n_long)
        return self.scene.mlab.plot3d(x, y, z, t)

    @on_trait_change('n_merid,n_long')
    def update_plot(self):
        x, y, z, t = curve(self.n_merid, self.n_long)
        self.plot.mlab_source.set(x=x, y=y,
                                  z=z, scalars=t)
```

The method `update_plot` is called when the `n_merid` or `n_long` attributes are modified, for instance through the UI. The `mlab_source` attribute of the plot object is used to modify the existing 3D plot without rebuilding it.

It is to be noted that the full power of the Mayavi library is available to the user in these dialogs. This is an extremely powerful feature.

Embedding mayavi in a wxPython application

Since TraitsUI provides a wxPython backend, it is very easy to embed Mayavi in a wxPython application. The previous TraitsUI code example may be embedded in a wxPython application:

```
import wx
from mlab_model import MyModel

class MainWindow(wx.Frame):
    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id,
                                'Mayavi in Wx')
        self.mayavi = MyModel()
        self.control = self.mayavi.edit_traits(
                        parent=self,
                        kind='subpanel').control
        self.Show(True)

app = wx.PySimpleApp()
frame = MainWindow(None, wx.ID_ANY)
app.MainLoop()
```

Thus, mayavi is easy to embed in an existing application not based on traits. Currently traits supports both wxPython and Qt as backends. Since two toolkits are already supported, it is certainly possible to support

more, although that will involve a fair amount of work.

### Mayavi in envisage applications

Envisage is an application framework that allows developers to create extensible applications. These applications are created by putting together a set of plugins. Mayavi2 provides plugins to offer data visualization services in Envisage applications. The `mayavi2`

application is itself an Envisage application demonstrating the features of such an extensible application framework by assembling the Mayavi visualization engine with a Python interactive shell, logging and preference mechanisms, and a docked-window that manages layout each provided as Envisage plugins.

### Customization of mayavi

Mayavi provides a convenient mechanism for users to contribute new sources, filters and modules. This may be done:

- at a global, system-wide level via a `site_mayavi.py` placed anywhere on Python's `sys.path`,
- at a local, user level by placing a `user_mayavi.py` in the users `~/.mayavi2/` directory.

In either of these, a user may register new sources, filters, or modules with Mayavi's central registry. The user may also define a `get_plugins` function that returns any plugins that the `mayavi2` application should load. Thus, the Mayavi library and application are easily customizable.

### Headless usage

Mayavi also features a convenient way to create offscreen animations, so long as the user has a recent enough version of VTK (5.2 and above). This allows users to create animations of their data. Consider the following simple script:

```
n_step = 36
scene = mlab.gcf()
camera = scene.camera
da = 360.0/n_step
for i in range(n_step):
    camera.azimuth(da)
    scene.reset_zoom()
    scene.render()
    mlab.savefig('anim%02d.png' % i, size=(600,600))
```

This script rotates the camera about its azimuth and saves each such view to a new PNG file. Let this script be saved as `movie.py`. If the user has another script to create the visualization (for example consider the standard `streamline.py` example) we may run these to provide an offscreen rendering like so:

```
$ mayavi2 -x streamline.py -x movie.py -o
```

The `-o` option (or `--offscreen`) turns on the offscreen rendering. This renders the images without creating a user interface for interaction but saves the PNG images. The PNG images can be combined to create a movie using other tools.

We have reviewed the various usage patterns that Mayavi provides. We believe that this variety of use cases and entry points makes Mayavi a truly reusable 3D visualization tool. Mayavi is not domain specific and may be used in any suitable context. In the next section we discuss the secrets behind this level of reusability and the lessons we learned.

## Secrets and Lessons learned

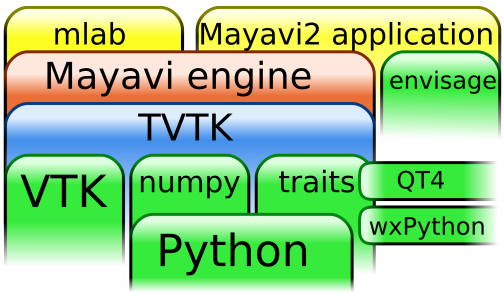
The techniques and pattern used to achieve maximal reusability in Mayavi are an application of general software architecture good practices. We will not review software architecture, although it is often underexposed to scientific developers, an introduction to the field can be found in [Gar94]. An important pattern in Mayavi's design is the separation between model and view, an introduction to which can be found in [Fow]. There are several contributing technical reasons which make Mayavi reusable:

- Layered functionality,
- Large degree of separation of UI from model,
- Object-oriented architecture and API,
- Scriptable from ground up,
- The use of Traits.

We look at these aspects in some detail in the following.

### Layered functionality

Mayavi is built atop layers of functionality, and a variety of different modules:




*Tool stack employed by Mayavi.*

At the lowest level of this hierarchy are VTK, numpy and Traits. The TVTK package marries VTK, numpy and Traits into one coherent package. This gives us the power of VTK with a very Pythonic API. TVTK is the backbone of Mayavi. Traits optionally depends on either wxPython or Qt4 to provide a user interface.

The core Mayavi engine uses TVTK and Traits. The `mayavi2` application and the `mlab` API use the Mayavi core engine to provide data visualization. The `mayavi2` application additionally uses Envisage to provide a plugin-based extensible application.

### Using Traits in the object model

The use of Traits provides us with a very significant number of advantages:

- A very powerful object model,
- Inversion of control and reactive/event-based programming: Mayavi and TVTK objects come with pre-wired callbacks which allow for easy creation of interactive applications,
- Forces a separation of UI/view from object model,
- Easy and free UIs:
  - Automatic user interfaces for wxPython and Qt4.
  - UI and scripting are well connected. This means that the UI automatically updates if the underlying model changes and this is automatically wired up with traits,
  - No need to write toolkit-specific code.

Traits allows programmers to think in very different ways and be much more efficient. It makes a significant difference to the library and allows us to completely focus on the object model.

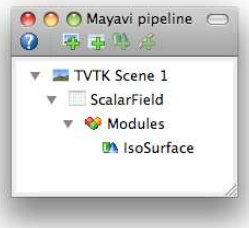
On the downsides, we note that automatically generated UIs are not very pretty. Traits provides methods to customize the UI to look better but it still isn't perfect. The layout of traits UI is also not perfect but is being improved.

### Object-oriented architecture

The object-oriented API of Mayavi and its architecture helps significantly separate functionality while enabling a great deal of code reuse.

- The abstraction layers of Mayavi allows for a significant amount of flexibility and reuse. This is because the abstraction hides various details of the internals of TVTK or VTK. As an example, the Mayavi `Engine` is the object central to a Mayavi visualization that manages and encapsulates the entirety of the Mayavi visualization pipeline.
- Ability to create/extend many Mayavi engines is invaluable and is the key to much of its reusability.
- All of Mayavi's menus (on the application as well as right-click menus) are automatically generated. Similarly, the bulk of the `mlab.pipeline` interface is auto-generated. Python's ability to generate code dynamically is a big win here.
- Abstraction of menu generation based on simple metadata allows for a large degree of simplification and reuse.
- The use of Envisage for the `mayavi2` application forces us to concentrate on a reusable object model. Using envisage makes our application extensible.

The `Engine` object is not just a core object for the programming model, its functionality can also be exposed via a UI where required. This UI allows one to edit the properties of any object in the visualization pipeline as well as remove or add objects. Thus we can provide a powerful and consistent UI while minimizing duplication of efforts, both in code and design.




*Dialog controlling the `Engine`: The different visualization objects are represented in the tree view. The objects can be edited by double-clicking nodes and can be added using the toolbar or via a right-click.*

In summary, we believe that Mayavi is reusable because we were able to concentrate on producing a powerful object model that interfaces naturally with numpy. This is largely due to the use of Traits, TVTK and Envisage which force us to build a clean, scriptable object model that is Pythonic. The use of traits allows us to concentrate on building the object model without worrying about the view (UI). Envisage allows us to focus again on the object model without worrying too much about the need to create the application itself. We feel that, when used as a visualization engine,

Mayavi provides more than a conventional library, as it provides an extensible set of reusable dialogs that allow users to configure the visualization.

Mayavi still has room for improvement. Specifically we are looking to improve the following:

- More separation of view-related code from the object model,
- Better and more testing,
- More documentation,
- Improved persistence mechanisms,
- More polished UI.

## Conclusions

Mayavi is a general-purpose, highly-reusable, 3D visualization tool. In this paper we demonstrated its reusable nature with specific examples. We also elaborated the reasons that we think make it so reusable. We believe that these general principles are capable of being applied to any project that requires the use of a user interface. There are only a few key lessons: focus on the object model, make it clean, scriptable and reusable; in addition, use test-driven development. Our technological choices (Traits, Envisage) allow us to carry out this methodology.

## References

[VTK] W. Schroeder, K. Martin, W. Lorensen, "The Visualization Toolkit", Kitware, 4th edition, 2006.

[M2] P. Ramachandran, G. Varoquaux, "Mayavi2 User Guide", http://code.enthought.com/projects/mayavi/docs/development/mayavi/html/

[Gar94] D. Garlan, M. Shaw, "An Introduction to Software Architecture", in "Advances in Software Engineering and Knowledge Engineering", Volume I, eds V.Ambriola, G.Tortora, World Scientific Publishing Company, New Jersey, 1993, http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf

[Fow] M. Fowler http://www.martinfowler.com/eaaDev/uiArchs.html