Customized Python Module Enables Analysis of Merging Binary Star Simulations within VisTrails

Joel E. Tohline, Jinghua Ge, Wesley Even, and Erik Anderson

By constructing a relatively simple, customized python module that plugs smoothly into an otherwise standard workflow within VisTrails, we have developed the capability to quantitatively analyze complex fluid flows in simulations of merging binary stars. The resulting visualization tool permits us to identify values of key rotational frequencies associated with such flows and how these frequencies change over time.

Scientific visualization tools being developed within the open-source community are steadily improving. In particular, they are providing a wider array of sophisticated probes for data analysis and a wider assortment of effective user-friendly interfaces. Such improved tools are making it easier for researchers in the computational sciences community – across many disciplines – to effectively analyze huge datasets by drawing on the human brain’s acute ability to sort through complex and time-varying visual patterns. The astrophysics group at LSU, for example, routinely use volume-rendering and ray-tracing algorithms in conjunction with animation techniques to examine the time-varying behavior of isodensity surfaces that arise from computational fluid dynamic (CFD) simulations of mass-transferring and merging binary star systems (see earlier Visualization Corner article: Nov/Dec 2007; v9n6076). While such analyses generally provide only a qualitative identification and assessment of structure within a given dataset, the insight gained from visual inspection can nevertheless be extremely valuable. For example, it was through visual inspection that the nonlinear development of triangular-, box-, and pentagonal-shaped tidal resonances were initially spotted in recent simulations.1,2

The astrophysics group at LSU has begun to incorporate VisTrails into its arsenal of scientific visualization and data analysis tools. The group was primarily drawn to VisTrails a few years ago because it provides a user-friendly “workflow” interface to the extensive vtk software library and because it automatically tracks the provenance of data analysis efforts.3 However, the ease with which VisTrails facilitates the insertion of home-grown analysis modules into an otherwise vtk-based workflow has significantly magnified our appreciation of the role that visualization tools can play in the quantitative assessment of results from large-scale simulations. In what follows, we first describe the vtk-based workflow that was initially constructed within VisTrails to view streamlines within each binary mass-transfer simulation. Then we describe the python module whose insertion into this workflow has permitted us to identify values of key rotational frequencies associated with such flows.

Terminology: One well-defined characteristic of a binary star system is its orbital period, P. If the stars are in circular orbit about one another, a binary system will appear to be stationary when viewed from a frame that is rotating with an angular frequency \( \Omega_{\text{frame}} = 2\pi/P \). When modeling mass-transferring binary star systems, we have found it advantageous to perform each CFD simulation on a cylindrical-coordinate grid that rotates with a frequency \( \Omega_0 = 2\pi/P_0 \), where \( P_0 \) is the binary system’s orbital period at the beginning of the simulation. As mass and angular momentum are transferred from one star to the other, however, the binary system’s orbital period – and associated value of \( \Omega_{\text{frame}} \) – is expected to vary throughout a simulation. As illustrated below, our new visualization tool can be used to measure \( \Delta\Omega = (\Omega_{\text{frame}} - \Omega_0) \) and, hence, the instantaneous orbital period \( P = P_0/(1 + P_0\Delta\Omega/2\pi) \) at any time during a simulation.
Base Workflow
Within VisTrails, we initially selected various vtk-based modules to (a) read in the simulation data, (b) outline the boundaries of the segment of the cylindrical grid containing the simulation data, (c) render two, nested isodensity surfaces to outline high-density (red) and low-density (blue) regions of the flow, and (d) draw eight separate streamlines to illustrate the nature of the binary mass-transfer flow. The chosen modules (see the “base workflow” depicted on the left-hand-side of Figure 1) and their intended functions are:

(a) Read simulation data – vtkPLOT3DReader was used to read in one file containing the (x,y,z) coordinate location of every vertex on our 3D cylindrical coordinate mesh, and a separate file containing the mass-density (scalar) and momentum density (3D vector) of the fluid at every grid vertex.

(b) Outline cylindrical domain boundary – as shown, we enlisted vtkStructuredGridOutlineFilter, vtkPolyDataMapper, & vtkActor.

(c) Define isodensity surfaces – the “Red_contour” and “Blue_contour” module groups each contain vtkContourFilter, vtkDataSetMapper, vtkProperty, & vtkActor.

(d) Draw_Streamlines group – as illustrated by the “Draw_Streamline” inset of Figure 1, each of 8 separate module groups uses vtkStreamLine, vtkTubeFilter, vtkDataSetMapper, vtkProperty, vtkActor, vtkSphereSource, vtkPolyDataMapper, & vtkLODActor. Streamline lengths are set by feeding a common “Propagation_Time” into all 8 module groups.

Output from the various “Actors” in steps (b) – (d) is rendered in a composite scene using vtkRenderer as viewed by an observer located at a position specified through vtkCamera and, finally, this scene is directed to the VisTrails interactive spreadsheet via the module VTKCell.

In this initially constructed “base workflow,” the 3D vector field representing the momentum density distribution was piped from the vtkPLOT3DReader module directly into each of the 8 “streamlines” module groups. This base workflow – assembled using generically available vtk modules – allowed us to examine the behavior of streamlines in our binary mass-transfer simulations, but only from the frame of reference, \( \Omega_b \), in which each simulation was originally performed (see rendered image “B” labeled \( \Delta \Omega = 0.00 \) in Figure 2).

Customized Python Module
In order to make it possible for us to examine the properties of binary mass-transfer flows from reference frames having a range of different angular frequencies of rotation, \( \Omega_{\text{frame}} = (\Omega_b + \Delta \Omega) \), we wrote a python-based module – called SwitchCoord – for insertion into the base VisTrails workflow described above. Our resulting customized VisTrails workflow is displayed on the right-hand-side of Figure 1. It differs very little from the base workflow. The complete python source code from our customized module has been printed in an accompanying sidebar to this article. The segment of the code that performs the required “physics analysis” is short and straightforward.

After assigning variable names to the data arrays acquired from the vtkPLOT3DReader module (“pcoords” is a tuple that identifies the Cartesian-based coordinate location of each grid vertex, “density” is a scalar that specifies the mass-density and “momentum” is a tuple that specifies the values of the cylindrical-coordinate-based vector momentum at each grid vertex), the SwitchCoord module performs the following operations at each grid vertex:

- Converts (x,y) Cartesian to (R,\( \phi \)) cylindrical coordinates.
- If the density is greater than “minp,” divides momentum components by density to obtain velocity components; otherwise sets velocity components to zero.
- Shifts azimuthal velocity component \( v_\phi \) to a new, rotating frame of reference by adding \( R * \Delta \Omega \).
- Converts cylindrical velocity components to Cartesian velocity components.
• Normalizes velocities to the maximum velocity “maxnorm” found across the domain where densities are greater than “minp.”

The output ports on SwitchCoord are designed to provide access to the same type of structured arrays as are generated by vtkPLOT3DReader. But in our customized VisTrails workflow that includes SwitchCoord (as depicted on the right-hand-side of Figure 1) the 3D vector field being piped into each of the 8 “Draw_Streamline” module groups represents the fluid’s velocity distribution as viewed from the rotating frame of reference that is specified by the floating-point scalar, “Omega_frame.”

Interpretation of Results

Figure 2 displays three-dimensional renderings of the flow from one of our binary mass-transfer simulations as generated by our customized VisTrails workflow, assuming five different frame rotation frequencies, as specified by \( \Delta \Omega \). We note that, aside from the labeling of \( \Delta \Omega \) values under each image, Figure 2 was obtained by simply taking a screenshot of the VisTrails interactive spreadsheet. The spreadsheet feature of VisTrails has proven to be extremely useful in this analysis because it facilitates the side-by-side comparison of scenes that have been rendered using different parameter values. And although it cannot be demonstrated here in the print medium, VisTrails allows the user to zoom, pan, and interactively rotate all the 3D rendered scenes simultaneously.

We would like to determine which value of \( \Delta \Omega \) provides the best measure of the true orbital period of the binary star system. As expected, for all five choices of \( \Delta \Omega \), the highest velocities (marked by the longest streamlines) are found along the relatively low-density mass-transfer stream that connects the two stars. Material from the donor star (in the lower half of each rendered image) flows toward its stellar companion, reaching supersonic velocities before impacting the companion. The component of motion perpendicular to the companion’s surface is terminated through an oblique shock front whose location is delineated by kinks in the pink, blue, and orange streamlines; motion transverse to the shock becomes orbital motion in a thick, low-density disk that surrounds the companion star.

For all five choices of \( \Delta \Omega \), the behavior of the flow in the vicinity of the mass-transfer stream very closely resembles the behavior predicted by Lubow & Shu\(^4\) over 30 years ago. On the right-hand-side of Figure 3 we have zoomed in on this region of the flow from our simulation, assuming \( \Delta \Omega = -0.041 \). This magnified image has been reoriented and the streamlines have been numbered to facilitate comparison with Figure 3 from Lubow & Shu, which has been reprinted here on the left-hand-side of our Figure 3. Rather than conducting a fully self-consistent 3D simulation – which was computationally impractical at the time – Lubow & Shu used a mathematical perturbation analysis to estimate what the flow should look like in the vicinity of the “L1” Lagrange point (identified in their diagram by the point where the two dot-dashed lines cross) as viewed from a frame of reference rotating with the correct instantaneous orbital frequency, \( \Omega_{\text{frame}} \). The close resemblance between our 3D simulation results in the vicinity of the L1 Lagrange point and the behavior predicted by Lubow & Shu provides one useful point of verification for our work.

Next, we note that the center of mass of each star should lie near the center of the highest density region (outlined by the nearly spherical, red isodensity surface) inside each star. When \( \Delta \Omega \) is assigned a value that properly identifies the frequency at which the centers of mass of the two stars are orbiting one another, we should see very little residual motion near the center of the donor star (i.e., the streamlines rendered in white and green in Figure 2 should be quite short). Furthermore, we expect that this residual motion should translate into concave streamline segments, mapping out simple circular motion about the center of the donor star. And when the correct value of \( \Delta \Omega \) is identified, we should expect the returning streamline nearest the mass-transfer stream (colored yellow) to remain inside the donor. With these ideas in mind, among the choices shown in Figure 2, we judge \( \Delta \Omega = -0.041 \).
Finally we note that as the pink and blue streamlines curve around the companion star, they extend outside of the companion’s disk (as outlined by the blue isodensity surface) in the rendered images with the most negative specified values of $\Delta \Omega$. These two streamlines appear to align most neatly with the distribution of material in the disk in the image that was rendered using $\Delta \Omega = + 0.02$. This suggests that there is a characteristic frequency associated with motion in the companion’s disk that is different from the binary orbital frequency.

**Summary**

A workflow constructed within VisTrails principally from publically available, vtk-based visualization modules has been extended to facilitate a non-standard data analysis effort by the LSU astrophysics group. This involved the straightforward incorporation of a relatively simple, customized python module into the VisTrails workflow. This enhanced scientific visualization tool is enabling us to tackle a complex data analysis task that previously has seemed to daunting to pursue.

Progress on this project has benefitted significantly from interactions with Claudio Silva, Juhan Frank, Patrick Motl, and Zach Byerly. This work has been supported, in part, by funding from the U.S. National Science Foundation (e.g., AST-0708551, DGE-0504507), U.S. Department of Energy, NASA (e.g., ATP-NNX07AG84G), and IBM; mass-transfer simulations were made possible through allocations of computing time on TeraGrid resources (TG-MCA98N043) at NCSA and LONI.

**References**


**Joel E. Tohline** is a professor at Louisiana State University. His research interests include astrophysics, computational fluid dynamics, and high-performance computing. Tohline has a PhD in astronomy from the University of California, Santa Cruz. He is a fellow of the AAAS, and a member of the International Astronomical Union, the American Astronomical Society, and the American Physical Society. Contact him at tohline@lsu.edu.

**Jinghya Ge** is a visualization consultant at the Center for Computation & Technology (CCT) at Louisiana State University. Her research interests include scientific visualization, computer graphics, and distributed computing. Ge has a PhD in computer science from the University of Illinois at Chicago. Contact her at jinghuage@cct.lsu.edu.

**Wesley Even** is an NSF/IGERT Fellow in the Department of Physics & Astronomy at Louisiana State University. His research focuses on the modeling of mass transfer in double-white-dwarf binary star systems. Contact him at weseven@physics.lsu.edu.

**Erik Anderson** is a research assistant and PhD candidate at the university of Utah. His research interests include scientific visualization, signal processing, computer graphics, and multi-modal visualization. Anderson has a BS in computer science and a BS in electrical and computer engineering from Northeastern University. Contact him at eranders@sci.utah.edu.
import core.modules.module_registry
from core.modules.vistrails_module import Module, ModuleError
import vtk, math
version="0.0.0"
name="SwitchCoord"
identifier="edu.lsu.switchcoord"

class SwitchCoord(Module):
    def compute(self):
        minp = self.getInputFromPort("min_density")
        Domega = self.getInputFromPort("Domaga")
        dataset=self.getInputFromPort("dataset")
        output = self.create_instance_of_type('edu.utah.sci.vistrails.vtk','vtkStructuredGrid')
        output.vtkInstance = vtk.vtkStructuredGrid()
        mydata=output.vtkInstance
        mydata.DeepCopy(dataset.vtkInstance)
        self.op(mydata, minp, Domega)
        self.setResult("changed_dataset", output)

        def op(self, mydata, minp, Domega):
            extent=mydata.GetExtent()
            pcoords = mydata.GetPoints().GetData()
            density = mydata.GetPointData().GetScalars("Density")
            momentum = mydata.GetPointData().GetVectors("Momentum")
            maxnorm = 0.0
            for i in range(0, mydata.GetNumberOfPoints()):
                [x, y, z] = pcoords.GetTuple3(i)
                [v1, v2, v3] = momentum.GetTuple3(i)
                r = math.sqrt(x*x + y*y)
                phi = math.atan2(y, x)
                if p < minp:
                    vx=vx=vy=vz=0
                else:
                    vr = _v1 / p
                    vphi = _v2 / (p) + r * Domega
                    vz = _v3 / p
                    vx = vr * math.cos(phi) - vphi * math.sin(phi)
                    vy = vr * math.sin(phi) + vphi * math.cos(phi)
                    norm = math.sqrt(vx*vx + vy*vy + vz*vz)
                    if norm > maxnorm:
                        maxnorm = norm
                        momentum.SetTuple3(i, vx, vy, vz)
            for i in range(0, mydata.GetNumberOfPoints()):
                [vx, vy, vz] = momentum.GetTuple3(i)
                vx = vx/maxnorm
                vy = vy/maxnorm
                vz = vz/maxnorm
                momentum.SetTuple3(i, vx, vy, vz)

        def initialize(*args, **keywords):
            reg=core.modules.module_registry.registry
            reg.add_module(SwitchCoord)
reg.add_input_port(SwitchCoord, "scalar_range", [core.modules.basic_modules.Float, core.modules.basic_modules.Float])
reg.add_input_port(SwitchCoord, "min_density", core.modules.basic_modules.Float)
reg.add_input_port(SwitchCoord, "Domega", core.modules.basic_modules.Float)
reg.add_input_port(SwitchCoord, "dataset", (reg.get_descriptor_by_name('edu.utah.sci.vistrails.vtk', 'vtkStructuredGrid').module) )
reg.add_output_port(SwitchCoord, "changed_dataset", (reg.get_descriptor_by_name('edu.utah.sci.vistrails.vtk', 'vtkStructuredGrid').module) )

def package_dependencies():
    import core.packagemanager
    manager = core.packagemanager.get_package_manager()
    if manager.has_package('edu.utah.sci.vistrails.vtk'):
        return ['edu.utah.sci.vistrails.vtk']
    else:
        return []
Figure 1