Assignment #4: 3D Steady Vector Field Visualization

Due on November 6th, before midnight

Goals:

The goals of this assignment include (1) understanding the characteristics of 3D steady vector fields, (2) extending the arrow plots and streamline placement techniques implemented for 2D vector fields (in your Assignment 3) to 3D and enable the different visualization styles of streamlines, and (3) implementing an interactive stream surface seeding and construction function.

A skeleton code in Python is provided to you to help you get started.

Requirements of Submission:

You should submit your source code **in .py format** and a writing report **in a separate file (pdf/docx)** via Canvas before the deadline. Please use the following naming convention to name your zip file:

*Assignment3_**FirstName_LastName.py***, *Assignment3_Report_**FirstName_LastName**.pdf (or .docx)*

Please DO NOT include the data sets in your submissions!!!!

Your report should include your answers to the writing questions **AND** high-quality images captured from your implemented program. For **each given 3D vector field**, please include the following in your report.

(1) An arrow plot (with clear arrow representation and little or no overlapping). Please provide the number of arrows shown in the visualization. **Not providing this will lead to the deduction of 5 points**.

(2) A streamline placement result using **both** tube-based AND stream-ribbon based visualizations (i.e., at least two images for streamline visualization). Please ***provide the number of streamlines and the selected seeding strategy (uniform or random)*** for each result. **Not providing this will lead to the deduction of 5 points.**

A difference in this assignment when compared to the previous assignments is that I will NOT provide detailed, step-by-step instruction on how to complete the individual programming tasks. Instead, I will point you to a few example python codes that implement similar functionality. You should now learn how to use those examples as references to help you figure out the correct way to complete the individual tasks.

Important Notice on the Use of ChatGPT in Programming Assignments:

While the utilization of ChatGPT or similar large language models is permitted for programming assignments, it is not allowed during exams. We urge students to exhaust their understanding and attempt solutions independently before resorting to these tools. Remember: the true value of this course is derived from the challenges you overcome, and the knowledge you gain in the process.

Transparency is crucial: If you choose to use ChatGPT or any other external resources, it is mandatory to clearly indicate so in your assignment report. You will not be penalized if you use ChatGPT. However, if you failed to report it and were caught, it would be considered plagiarism and your score will be zero. Note that it was prohibited to borrow answers and codes from students who already took this course.

Tasks:

1. Writing questions (20 points)

1.1 What needs to be considered when computing stream surfaces for 3D vector fields? **(10pts)**

1.2 How to perform LIC in 3D? Why is texture-based visualization not very effective for 3D vector fields? **(10 pts)**

2. Generate arrow plots (30 points)

Based on the similar idea and process as 2D arrow plot and an example 3D arrow plot shown in "arrows3d_Ex.py", complete the 3D arrow plot for general 3D vector fields. Your arrow plots should reveal the flow configuration with little occlusion and clutter.

Suggestion: To produce an effective arrow plot with less occlusion and clutter, you should consider using the `vtkMaskPoints` filter to generate a sparse set of samples where the arrows will be placed. See the instruction provided in Assignment 3 on how to use this filter.
Your arrow plot will be turned on and off by toggling the "Arrow plot" checkbox on the interface.

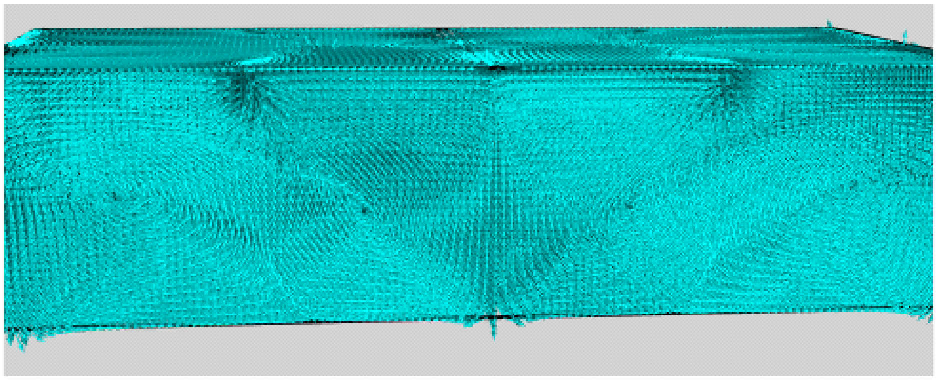


Figure 1. *Arrow plot of the bernard3D by placing an arrow at each grid point that leads to severe occlusion and clutter.*

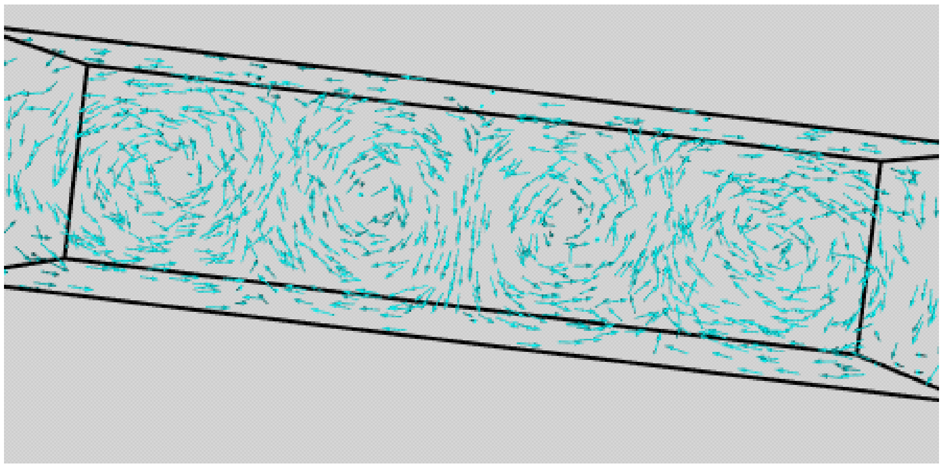


Figure 2. *Arrow plot of the bernard3D data with 1000 arrows positioned at some uniform samples, using the "UNIFORM_SPATIAL_BOUNDS" random mode of the* `vtkMaskPoints` *filter.*

3. Compute and visualize streamlines (50 points)

Next, you need to generate a number of seeds to compute a few streamlines for visualization. In this assignment, you are required to implement the following seeding and rendering strategies.

3.1 Seeds generation (20 points)

(1) **Extend the two seeding strategies** (i.e., uniform seeding and random seeding) that you have implemented in Assignment 3 for 3D streamline seeding. **(6 pts for each)**

(2) Now, let us implement a new seeding strategy by **specifying a seeding curve/line** in the data domain and uniformly place seeds along it **(8 points)**. You can look at the "StreamlinesWithLineWidget.py" (https://kitware.github.io/vtk-examples/site/Python/VisualizationAlgorithms/StreamlinesWithLineWidget/) on how to achieve that.

After generating the seeds, pass them to the `vtkStreamTracer` as how you did it in Assignment 3.

NOTE: This is the hardest task in the assignment! Do other tasks first if you are stuck. If you are unable to implement an interactive line widget, then try to earn partial points by implementing a static (non interactive) line seeding strategy that places seed point along a diagonal line (or horizontal/vertical line works too) across the data set. You can do this using **vtkLineSource** to pick any two points within the data set and place seed points on the line connected by these two points.

3.2 Create Sphere Actor (4 points)

Draw a small sphere at the location of each seeding point.

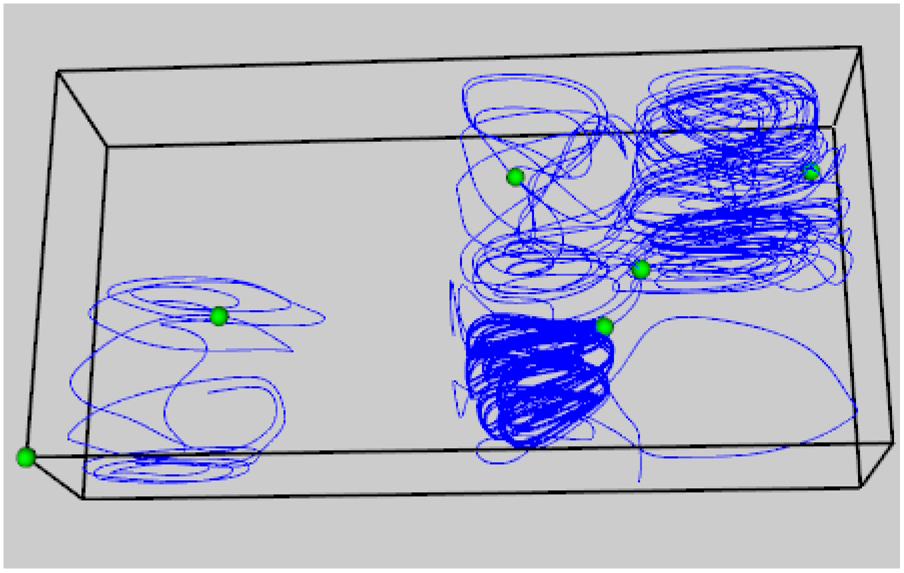


Figure 3. *An example of visualizing seed points.*

Refer to the arrow visualizations as the process is nearly identical with a few changes:

(1) use a **vtkSphereSource** instead of vtkGlyphSource or vtkArrowSource

(2) instead of using the data set (self.reader) as the InputData of the glyph, use seedPolyData (or whatever you use to store your seeding points poly data).

3.2 Rendering style (26 points)

(1) Visualize the computed streamlines using a **tube-based** representation **(9 points)**. See the "**officetube.py**" (https://gitlab.kitware.com/vtk/vtk/blob/cff62a106f99c9bac3d1bc4a4e449d28b7d94285/Examples/VisualizationAlgorithms/Python/officeTube.py) for an example on how to enable the tube-based rendering of the streamlines.

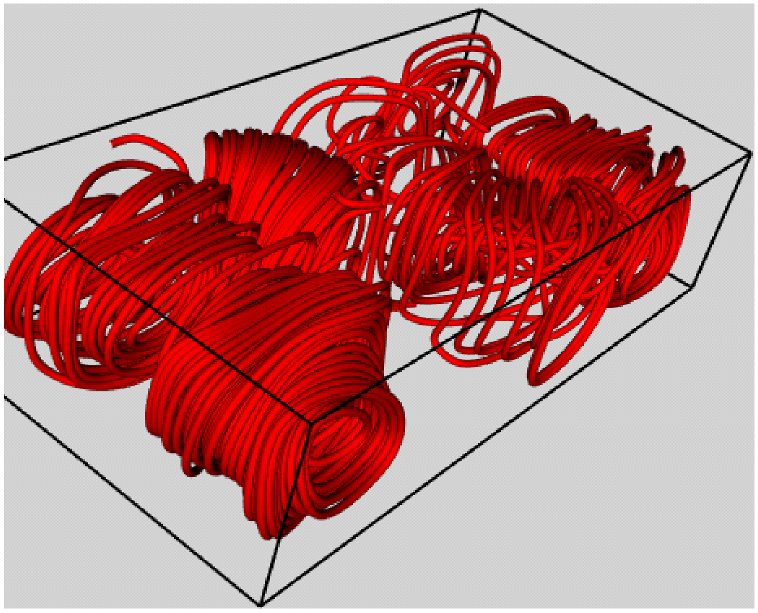


Figure 4. *An example tube-based representation of 10 randomly seeded streamlines for the bernard3D.*

(2) Visualize the computed streamlines using a **ribbon-based representation**. **(9 points)** Use the `vtkRibbonFilter` (see some examples in https://python.hotexamples.com/examples/vtk/-/vtkRibbonFilter/python-vtkribbonfilter-function-examples.html)

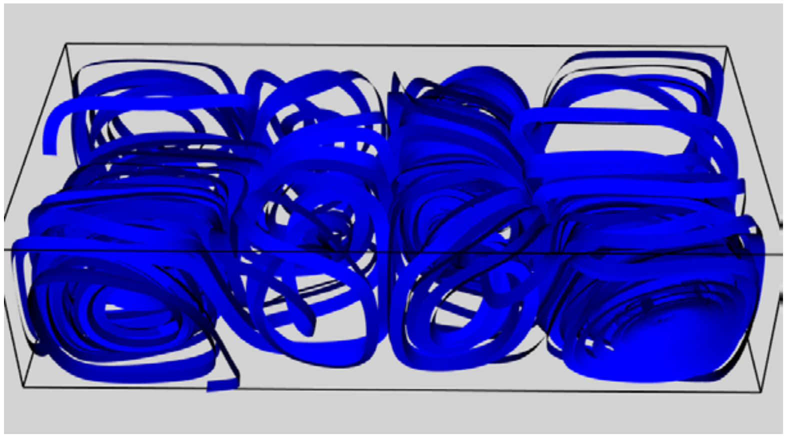
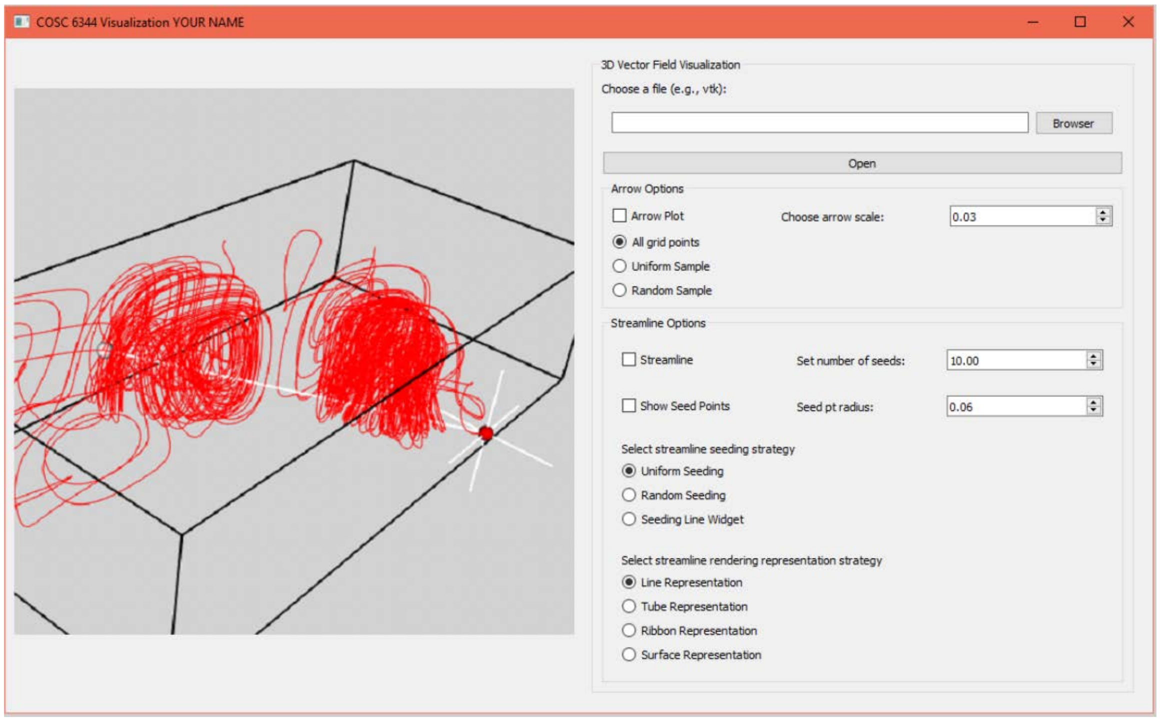


Figure 5. *An example ribbon-based representation of 10 randomly seeded streamlines for the bernard3D.*

Include an interface to allow the user to select among different seeding strategies and streamline rendering style.


COSC 6344 Visualization YOUR NAME

3D Vector Field Visualization

Choose a file (e.g., vtk):

Browser

Open

Arrow Options

☐ Arrow Plot — Choose arrow scale: 0.03

[✓ All grid points]
Uniform Sample
Random Sample

Streamline Options

☐ Streamline — Set number of seeds: 10.00

☐ Show Seed Points — Seed pt radius: 0.06

Select streamline seeding strategy
[✓ Uniform Seeding]
Random Seeding
Seeding Line Widget

Select streamline rendering representation strategy
[✓ Line Representation]
Tube Representation
Ribbon Representation
Surface Representation

Figure 6. *An example interface design for your reference. Feel free to design your own interface.*

(4). **Stream surface** placement and construction **(8 points)**

Use the above line widget and `vtkRuledSurfaceFilter` following the example (https://github.com/wildmichael/ParaView/blob/master/VTK/Examples/VisualizationAlgorithms/Python/streamSurface.py) to construct a good stream surface to depict each of the given 3D vector field.

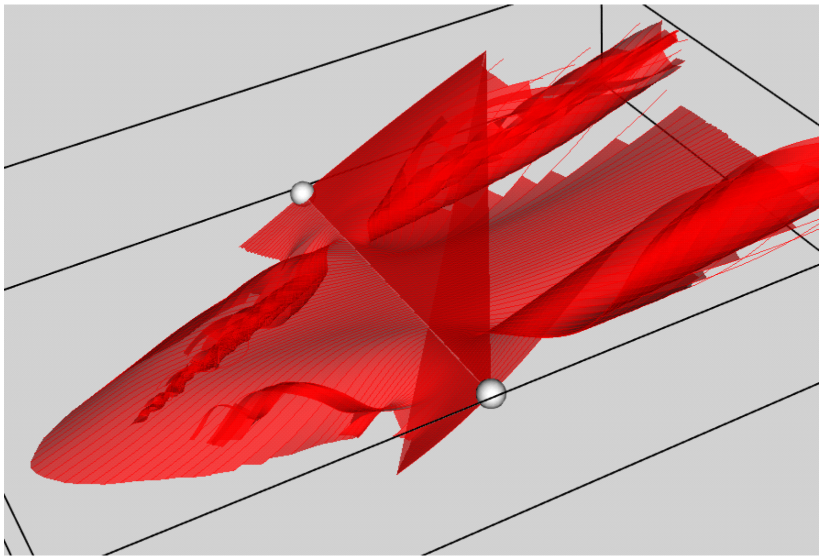


Figure 7. *An example surface for the delta wing data.*

Add an interface (e.g., a checkbox) to enable the interaction and construction of the stream surface.

Note that using the `vtkRuledSurfaceFilter` to stitch the neighboring streamlines to form a stream surface **does not always produce the correct surface** (as shown in the above example). This is fine for this assignment.

Note that *when the user switches to different visualization options (by toggling on and off checker boxes or selecting through radio buttons), the previous visualization should be cleaned out before showing the new visualization. Overlapping with the previous visualization will cause confusion and misunderstanding of the data. And, we will deduce points for it.*