VTK Lesson 03

Callbacks, Glyphing and Picking

112793 Cláudia Seabra, 131224 Francisco Costa
Information Visualization, 2025/26
MSc Informatics Engineering, University of Aveiro

Abstract

This lesson introduces advanced interaction and data representation techniques in the Visualization Toolkit (VTK), focusing on glyph-based visualization, object picking, unstructured grids, and the association of scalar and vector data.

This lesson explores how geometric glyphs can be used to represent multidimensional data, how users can interactively select elements within a 3D scene, and how unstructured datasets can be visualized and enhanced with vector and scalar attributes. Additionally, callback mechanisms are employed to dynamically respond to user actions, reinforcing the interactive nature of scientific visualization.

Objectives

The main objective of this lesson is to understand how VTK supports interactive visualization and the representation of complex datasets. The specific goals are:

- To explore glyphing techniques for visualizing vector and point-based data;
- To implement object picking and associate callbacks to user interactions;
- To display selected data attributes dynamically within the rendering window;
- To understand the structure and visualization of unstructured grids;
- To associate scalar and vector data with grid points and visualize them using glyphs;
- To compare different vector visualization techniques such as glyphs and hedgehogs.

Visualization Solution

Glyphing

Glyphing is a visualization technique where geometric shapes (glyphs) are placed at data points to represent additional information such as direction, magnitude, or orientation. In this lesson, the `vtkGlyph3D` class is used to place cone glyphs at each vertex of a sphere's polygonal model.

The sphere provides the input data (glyph positions), while the cone acts as the glyph source:

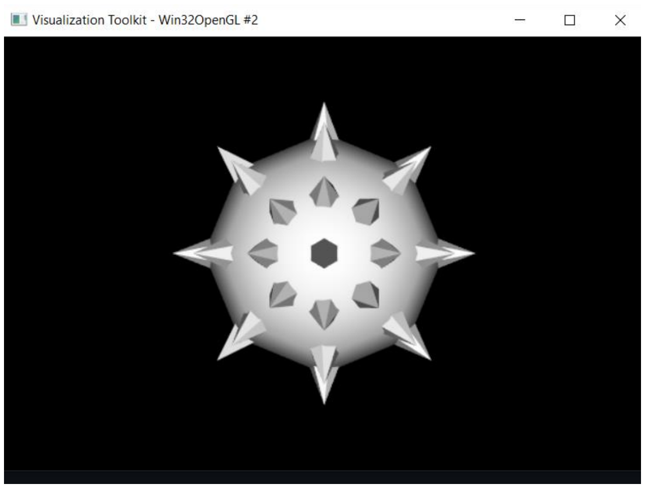
```
glyph = vtkGlyph3D()
glyph.SetSourceConnection(coneSource.GetOut
putPort())
glyph.SetInputConnection(sphereSource.GetOu
tputPort())
```

The `SetScaleFactor()` method controls the size of the glyphs, while

`SetVectorModeToUseNormal()` aligns each glyph with the normal vector of the surface at that point. As a result, the cones are oriented radially outward from the sphere, visually encoding surface orientation.

By modifying the sphere's theta and phi resolutions using `SetThetaResolution()` and `SetPhiResolution()`, the number of glyphs increases or decreases accordingly. Higher resolutions produce a denser and smoother distribution of cones, while lower resolutions result in fewer, more sparsely placed glyphs.

To achieve the specific visual outcome shown in the image below, the implementation relies on the coordnation of two separate actors and management of the underlying geometry.



The final image is composed of two distinct objects rendered together:

- A sphere: A `vtkSphereSource` is mapped to a `sphereActor`, so to provide a solid gray body in the center.
- The Glyphs (Cones): The `vtkGlyph3D` filter takes the same sphere as an input but replaces its vertices with cones. These are mapped to a `glyphActor`.

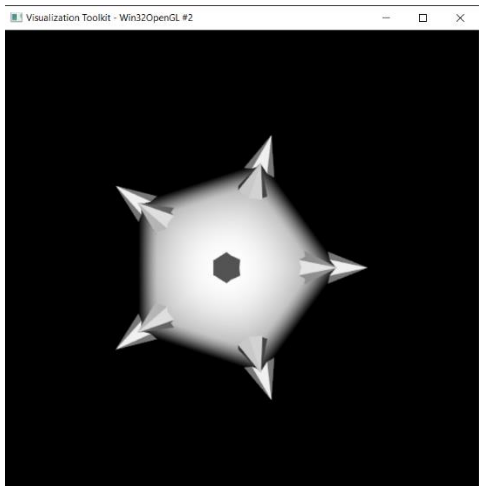
The "look" of the spikes is controlled by the sphere's resolution. By setting `SetThetaResolution(8)` and `SetPhiResolution(8)`, we reduced the number of points on the sphere. This ensures the cones do not overlap and creates the symmetrical, spaced-out pattern seen in the reference.

`SetVectorModeToUseNormal()` is critical. It's used to tell VTK to read the surface normals of the sphere and rotate each cone so its tip points directly away from the center of the sphere.

Impact of Resolution

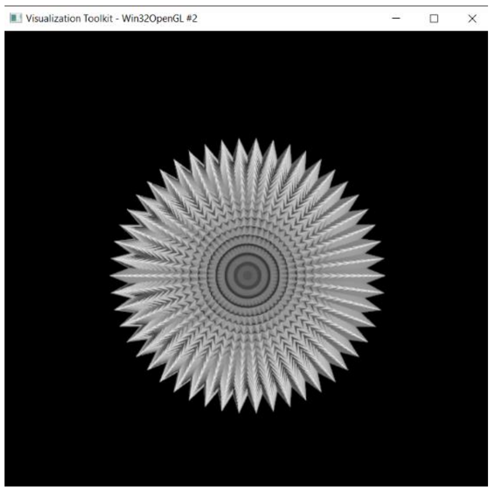
By modifying the sphere's theta and phi resolutions using `SetThetaResolution()` and `SetPhiResolution()`, the number of glyphs increases or decreases accordingly.

Decreasing Resolution (5x5):



The number of cones decreases significantly, making the sphere look like a low-polygon object. There is more visible space between the bases of the cones, as there are fewer vertices available to host a glyph.

Increasing Resolution (50x50):



The sphere becomes crowded with cones, hiding the sphere's surface entirely.

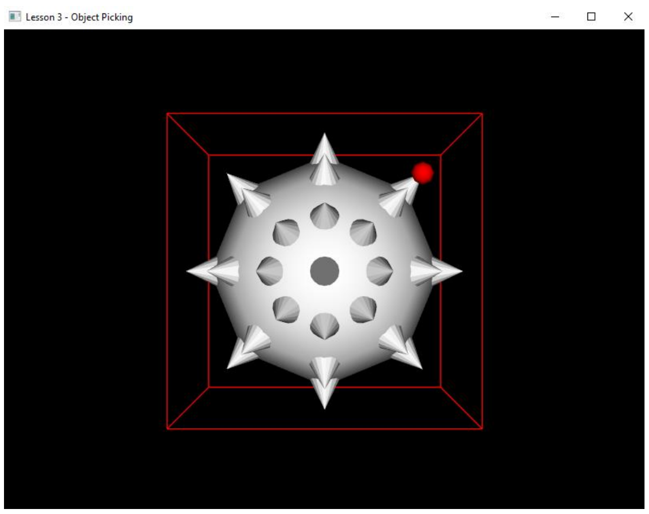
Object Picking

To enable user interaction, a `vtkPointPicker` is added to the render window interactor. This allows the user to select points on the model by pressing the predefined P key.

```
myPicker          =          vtkPointPicker()
iren.SetPicker(myPicker)
```

A callback function is associated with the picker to retrieve the coordinates of the selected point using `GetPickPosition()`. These coordinates are printed to the command window, providing immediate feedback about the selected location.

Additionally, a small sphere actor is created to visually mark the selected point. Initially invisible, the sphere becomes visible after a point is picked and is repositioned using the actor's `SetPosition()` method. This significantly improves the clarity of the interaction by providing a visual cue within the 3D scene.



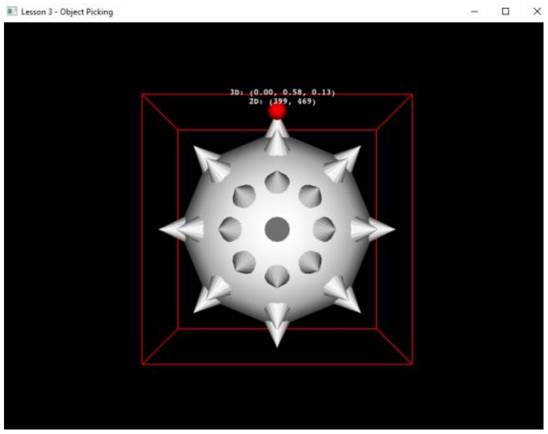
Display of Coordinates

The callback mechanism is extended to display the coordinates of the selected point directly within the renderer. This is achieved using a `vtkTextMapper` and a `vtkActor2D`, which allow 2D text to be overlaid on the rendering window.

The callback performs the following steps:

1. Retrieves the selected point's 3D coordinates (`GetPickPosition`);
2. Retrieves the corresponding 2D pixel coordinates (`GetSelectionPoint`);
3. Updates the text content of the `vtkTextMapper`;
4. Positions the text actor near the selected point;
5. Makes the text actor visible.

The text formatting is customized using the text mapper's properties, including centered alignment, bold style, and the Courier font. This approach demonstrates how numerical data can be visually integrated into interactive 3D scenes.



Unstructured Grid

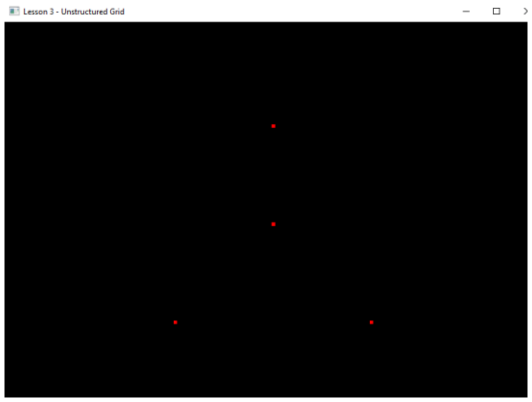
An unstructured grid is analyzed using the provided `ugrid.py` example, which initially visualizes a single tetrahedral cell. The code is modified to display only the grid's vertices by changing the cell type from `VTK_TETRA` to `VTK_VERTEX`.

The actor's properties are adjusted to improve visibility:

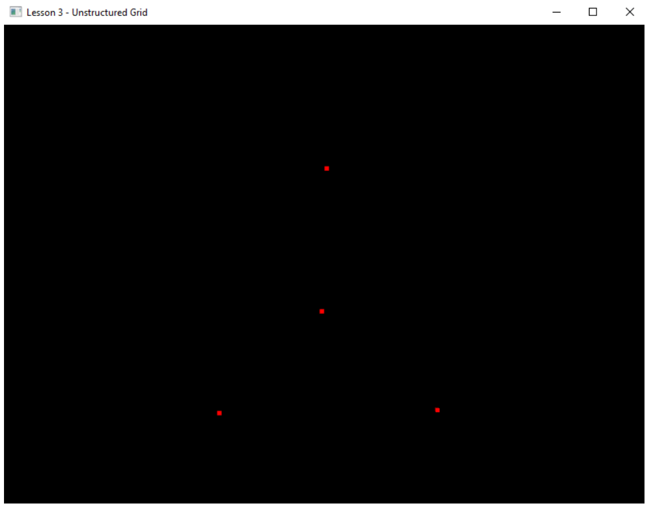
```
UgridActor.GetProperty().SetColor(1, 0, 0)
UgridActor.GetProperty().SetPointSize(5)
```

This modification highlights the individual points of the grid, providing a clearer understanding of how unstructured datasets can represent discrete geometric entities.

Top-View:



Side-View:



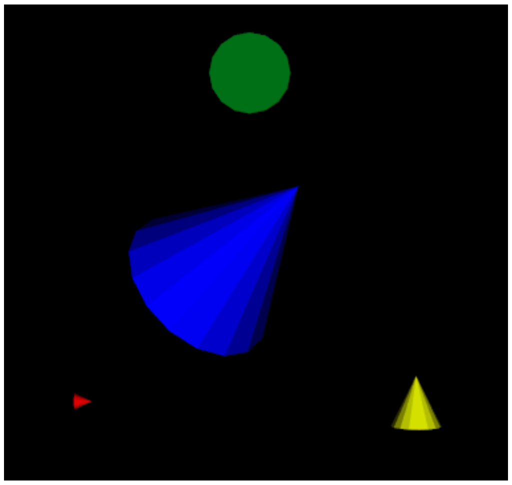
Scalar Association to Vectors and Grids

Vector data is associated with each vertex of the unstructured grid using a `vtkFloatArray` with three components. Each vector is assigned using the `InsertTuple3()` method, and the array is linked to the grid via `SetVectors()`.

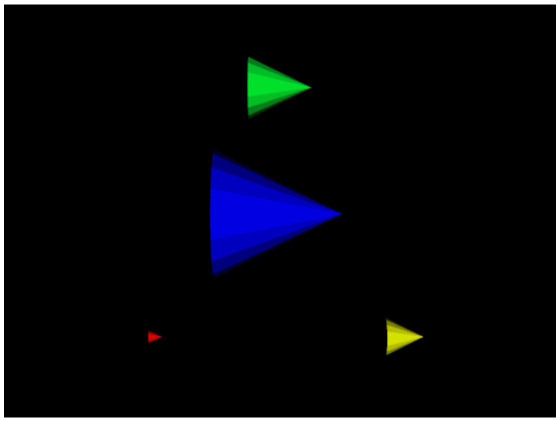
A cone glyph is then used to visualize the vector field, with each cone oriented according to the associated vector. This results in a clear directional representation at each grid point.

Subsequently, scalar values between 0 and 1 are associated with the grid points using another `vtkFloatArray` with a single component. When these scalars are assigned via `SetScalars()`, VTK automatically maps them to color values using the active lookup table. The cones are therefore colored according to their scalar magnitude.

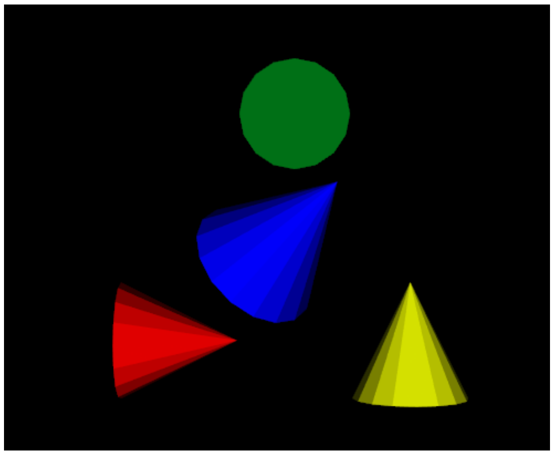
By enabling or disabling orientation and scaling options in `vtkGlyph3D`, different visualization outcomes are observed, illustrating how glyph properties can encode multiple data dimensions simultaneously.



With orientation disabled, all the cones face the same way:



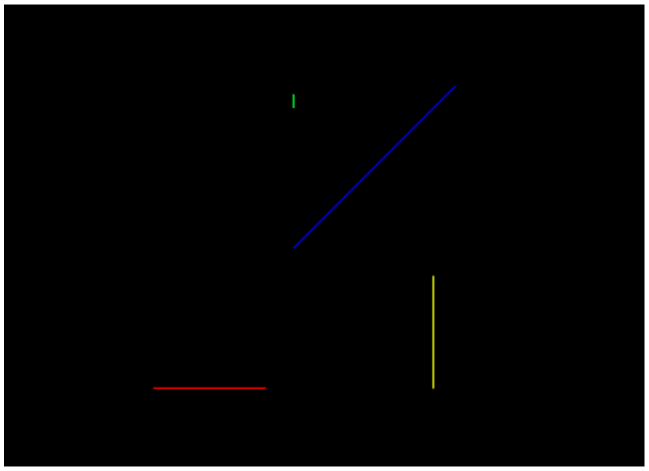
By disabling the scale mode, all the cones end up the same size:



HedgeHog

As an alternative to glyph-based visualization, the `vtkHedgeHog` class is explored. This class represents vector data as line segments extending from each point in the direction of the vector.

When applied to the same vector data used in the glyph example, the hedgehog visualization produces a simpler, less visually complex representation. While it lacks the expressive geometry of glyphs, it is effective for quickly conveying vector direction and relative magnitude, particularly in dense datasets.



Conclusion

Lesson 3 significantly enhances the understanding of interactive and multidimensional visualization in VTK. Through glyphing, picking, and callback mechanisms, the lesson demonstrates how geometric representations and user interaction can be combined to explore complex datasets. The use of unstructured grids, along with scalar and vector associations, illustrates how VTK supports advanced scientific visualization workflows. Finally, the comparison between glyphs and hedgehogs highlights the importance of choosing appropriate visualization techniques depending on the data and analytical goals.