Seamless integration across developer ecosystems with python and wasm in VTK

Jaswant Panchumarti [1], Sebastien Jourdain [1], Berk Geveci [1], Aashish Chaudhary [1]
[1]Kitware, Inc.

Introduction

- The Visualization Toolkit (VTK) underpins advanced **3D scientific visualization**.
- Currently, VTK's Python wrappers retain original C++ conventions, **making it very un-pythonic**.
- Middleware like **PyVista** and **Vedo** work hard to **minimize the boilerplate** needed to use VTK from Python.
- **Trame**, a python package helps to **build client-server web applications** that visualize datasets remotely or locally in a browser using pure Python.
- However, the 3D visuals generated with **client-side rendering in Trame are a bit different from VTK native** due to implementation differences in VTK.js

Challenges

- Need VTK's python bindings to **play well with NumPy, Xarray and Pythonic conventions**.
- VTK.wasm **OpenGL code needs to comply with WebGL2** standards set by browsers.
- Need **native Pythonic API** for VTK.
- Need **automated serialization** infrastructure to enable local rendering with VTK.wasm and trame.

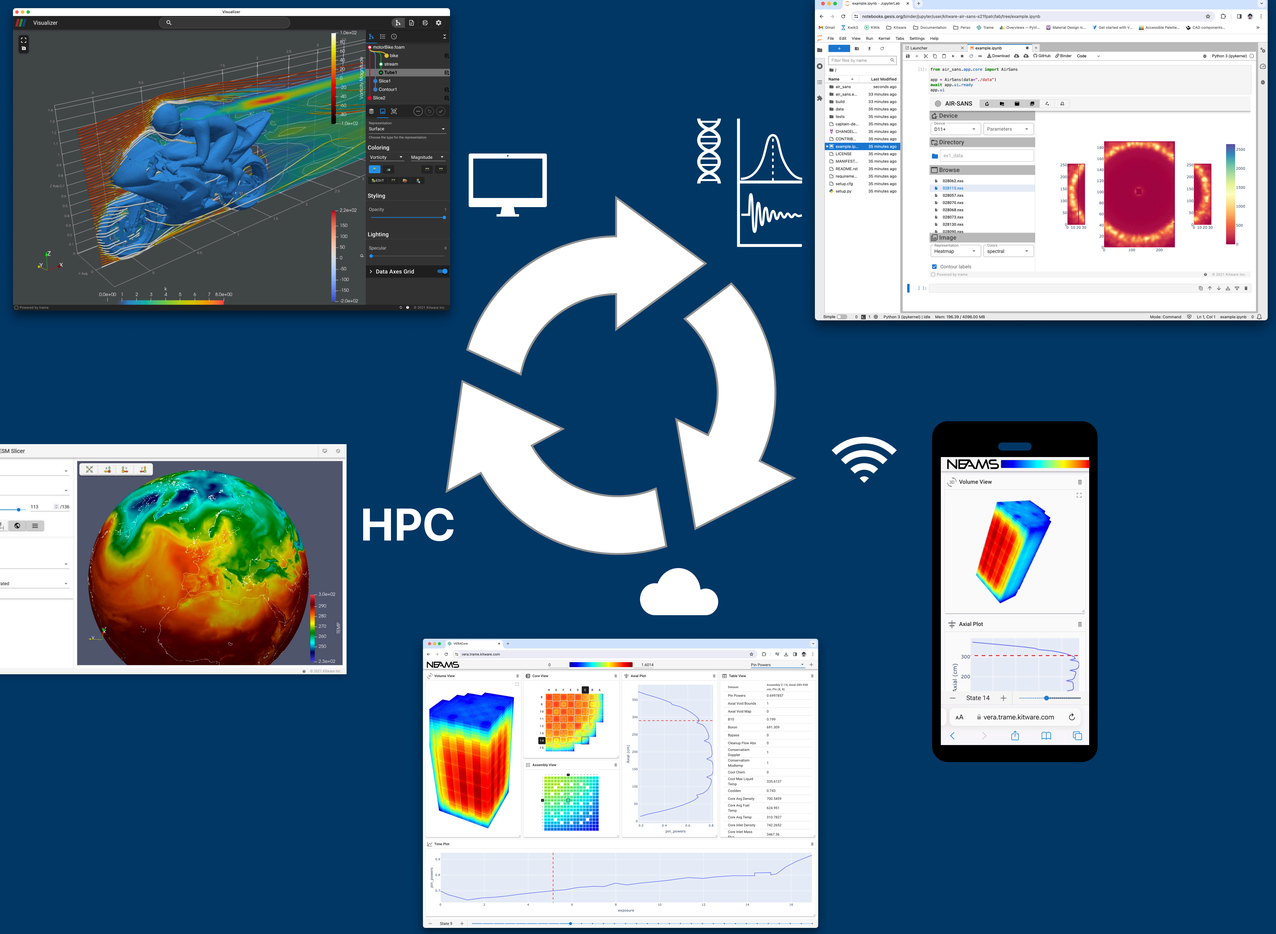
Approach

- Improve **VTK wrapping infrastructure** to better serve Python and WASM serialization.
- Support **keyword arguments in the constructor** for compact object initialization.
- Use **snakecase names for the properties** of wrapped VTK objects. These are internally mapped to the C++ setter and getter counterparts.
- Transparently **interchange between NumPy, Xarray, and VTK native array** objects using NumPy's array interface.
- Minimize boilerplate code to **connect VTK filters in a pipeline with new shorter syntax**.
- Ensure that the constantly evolving VTK source code **compiles successfully with Emscripten** compiler.
- **Automated serialization** of VTK objects for WASM rendering with trame.

Conclusion

The work done on the wrapping infrastructure of VTK has improved the usability and capabilities of 3D visualization and analysis for the Python and Web community.

Scientific visualization in Python with *trame* and *VTK.wasm* **everywhere**



Object Initialization

```
tracer = vtkStreamTracer()
tracer.SetInitialIntegrationStep(0.02)
tracer.SetTerminalSpeed(1400)
```

```
tracer = vtkStreamTracer(
    initial_integration_step=0.02
    terminal_speed=1400
)
```

Properties

```
renderer.SetBackgroundColor(0.4, 0.3, 0.8)
actor.GetProperty().SetOpacity(0.4)
```

```
renderer.background_color = (0.4, 0.3, 0.8)
actor.property.opacity = 0.4
```

VTK pipelines with operator '»'

```
tracer = vtkStreamTracer()
tracer.SetInputDataObject(grid)
tracer.SetIntegratorTypeToRungeKutta45()
tracer.SetIntegrationDirectionToBoth()
tracer.SetInitialIntegrationStep(0.2)
tracer.SetMaximumPropagation(3.2)

sphere = vtkSphereSource()
sphere.SetThetaResolution(16)

tracer.SetSourceConnection(sphere.GetOutputPort())

tube = vtkTubeFilter()
tube.SetNumberOfSides(3)
tube.SetRadius(0.00383538)
tube.SetInputConnection(tracer.GetOutputPort())

mapper = vtkPolyDataMapper()
mapper.SetInputConnection(tube.GetOutputPort())

actor = vtkActor()
actor.SetMapper(mapper)
```

```
tracer = vtkStreamTracer(
    integrator_type=vtkStreamTracer.RUNGE_KUTTA45,
    integration_direction=vtkStreamTracer.BOTH,
    initial_integration_step=0.2,
    maximum_propagation=3.2)

sphere = vtkSphereSource(theta_resolution=16)

grid >> select_ports(0, tracer)
sphere >> select_ports(1, tracer)

actor = vtkActor()
actor.mapper = (tracer
    >> vtkTubeFilter(
        number_of_sides=3,
        radius=0.00383538)
    >> vtkPolyDataMapper()).last
```

Trame example

```
@TrameApp()
class Cone:

    @change("resolution")
    def on_resolution_change(self, resolution, **kw):
        self._cone.resolution = resolution
        self.ctrl.update()

    def _build_ui(self):
        with SinglePageLayout(self.server) as layout:
            with layout.toolbar as toolbar:
                v3.VSlider(
                    v_model=("resolution", 6),
                    min=3, max=60, step=1,
                )
                v3.VBtn(icon="mdi-crop-free",
                        click=self.ctrl.r_camera)

            with layout.content:
                v = VtkLocal(self._rw) # WASM
                self.ctrl.update = v.update
                self.ctrl.r_camera = v.reset_camera
```

The Visualization ToolKit

The Trame Guide