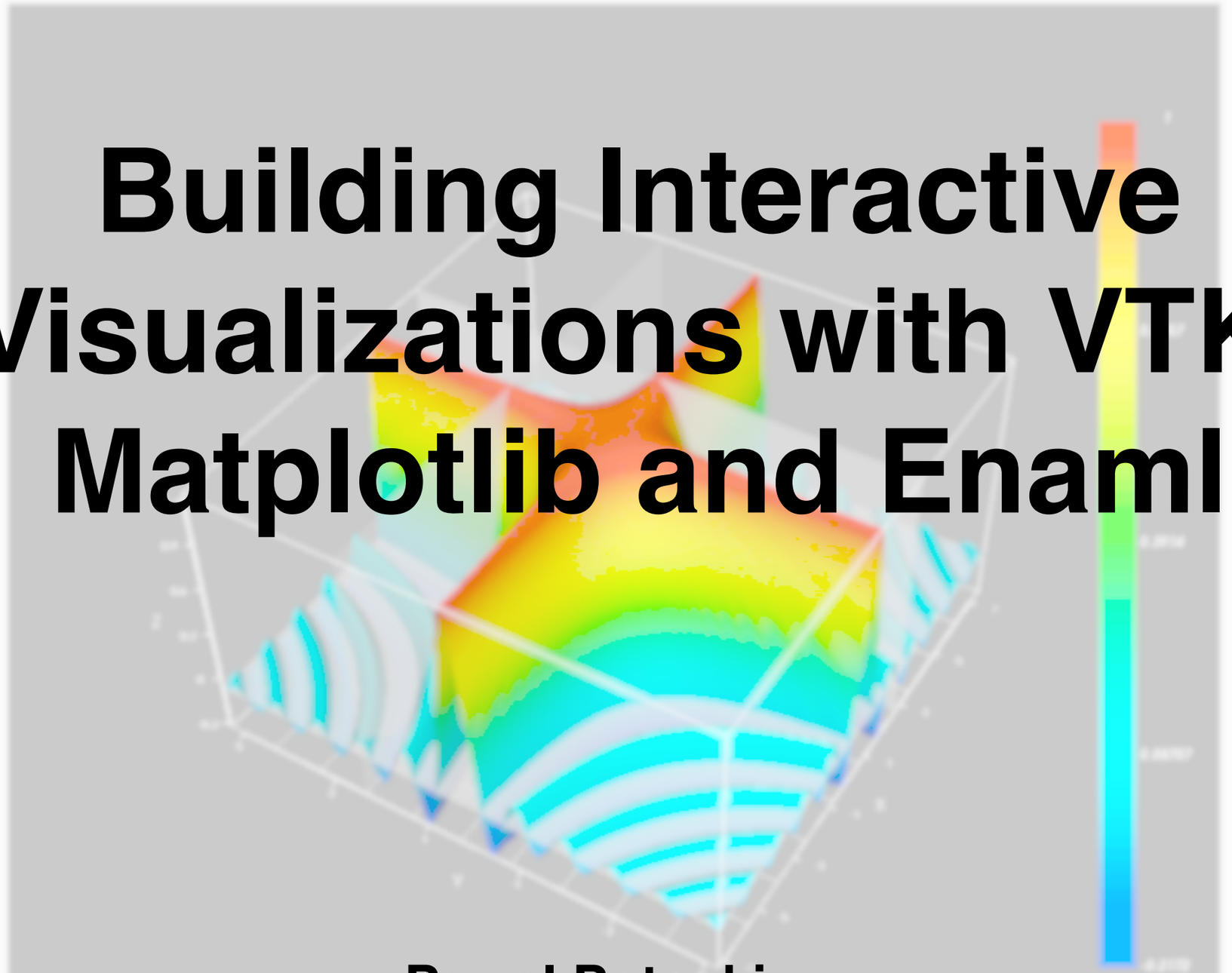
Building Interactive Visualizations with VTK, Matplotlib and Enaml



Pawel Potocki

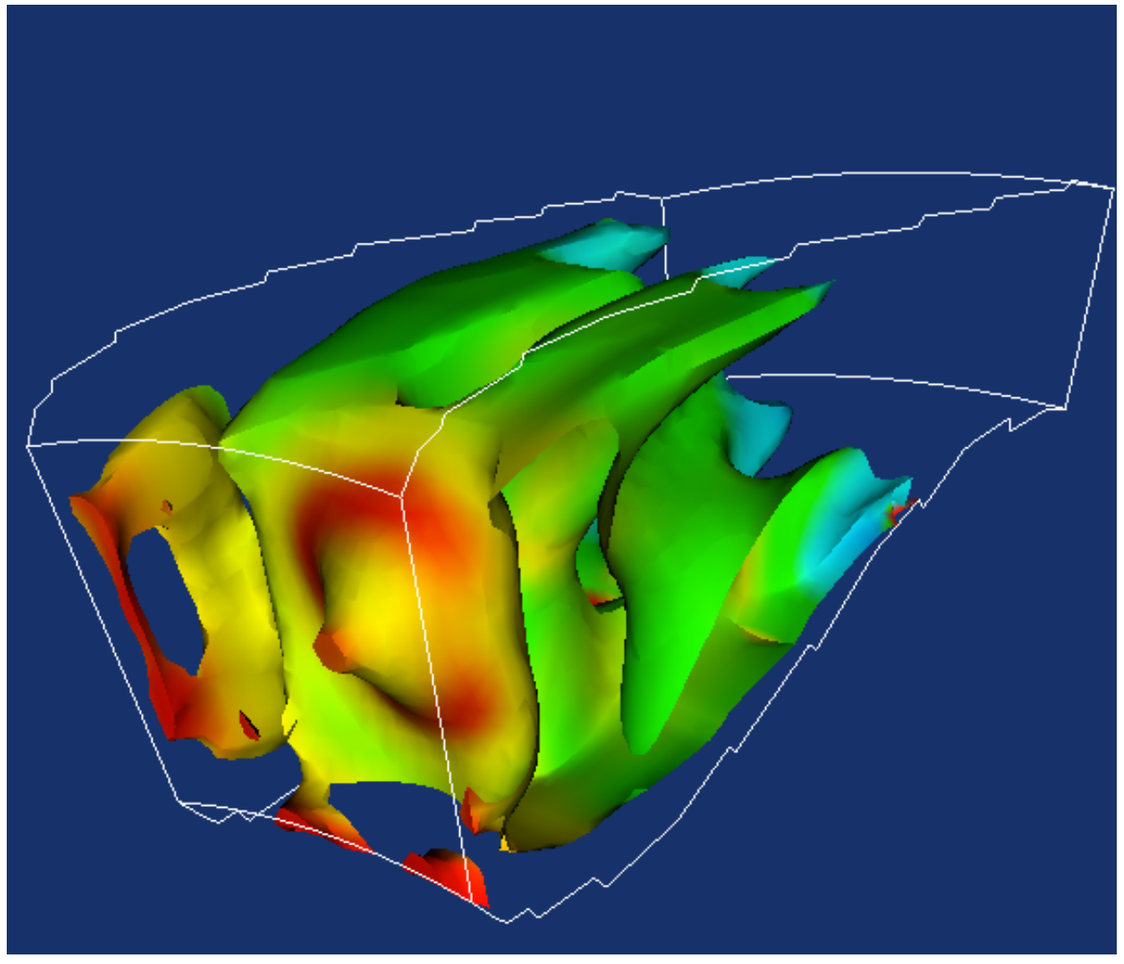
PyData NYC 2015

Introduction

- Pawel Potocki
- JPMorgan Commodities EMM Technology
- Data Visualization and Analysis in late 1990 and early 2000 at JPMorgan with C++ and Java
- Last few years of heavy Python development
- Data Visualization important part of work

Tutorial Content

- Visualization tools in Python - overview
- UI Toolkits in Python - looking for best choice
- Enaml - UI toolkit explained
- Enaml - features overview - examples
- VTK visualization pipeline explained
- VTK - building with Enaml - examples
- Matplotlib charting overview
- Matplotlib - building with Enaml - examples
- VTK and Matplotlib interaction with Enaml - examples
- Closing notes
- Resources

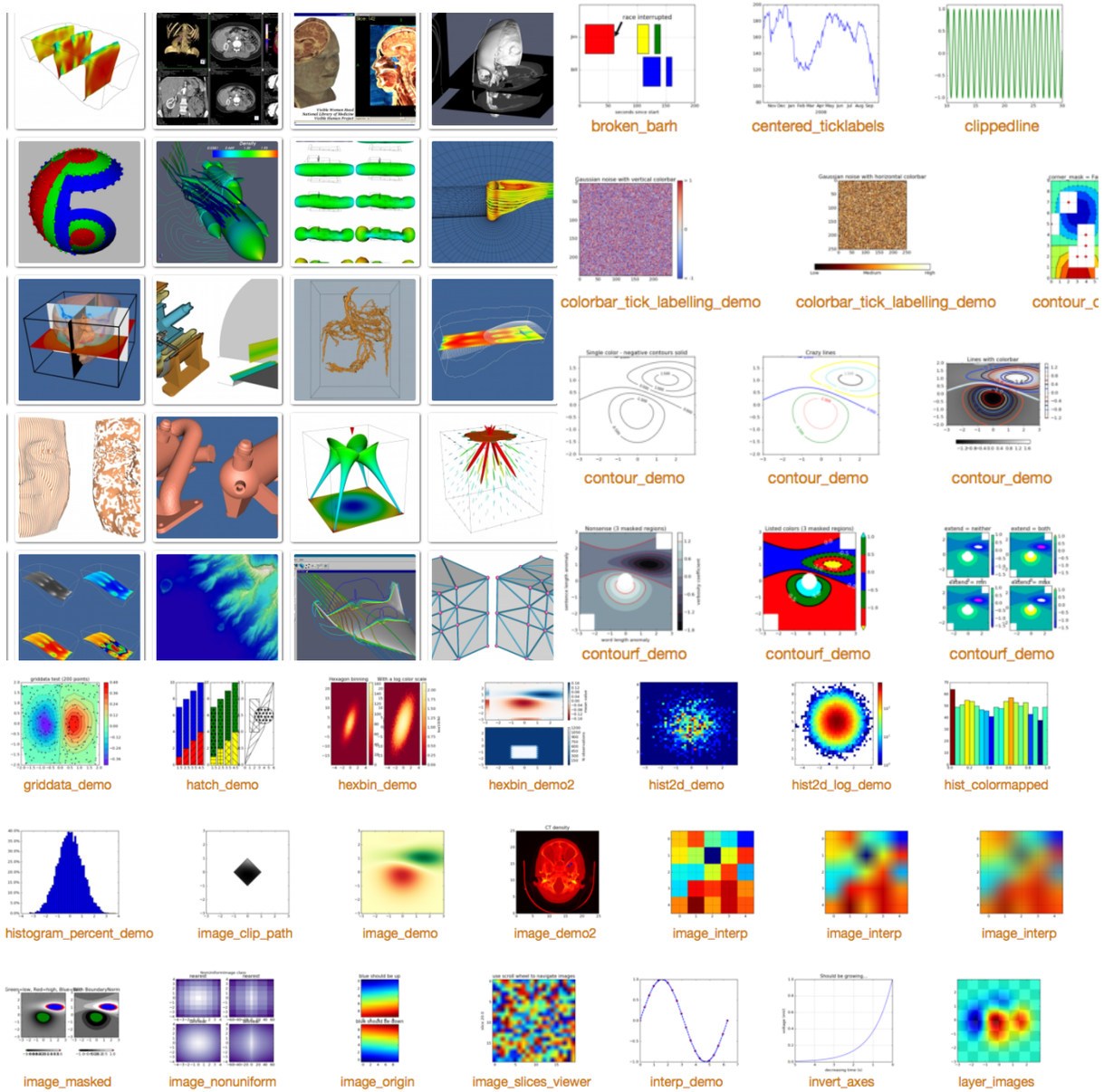


Why building interactive visualizations

- There are many UI Web frameworks based on HTML5 and Canvas - like Bokeh
- Using Jupiter notebook (ipython)
- Desktop apps still offer advantages over web apps
- In many cases we need to inspect data by visualization
- Building interactive apps quickly in business environment is crucial to help making right decision
- Quality and easy of operation is important

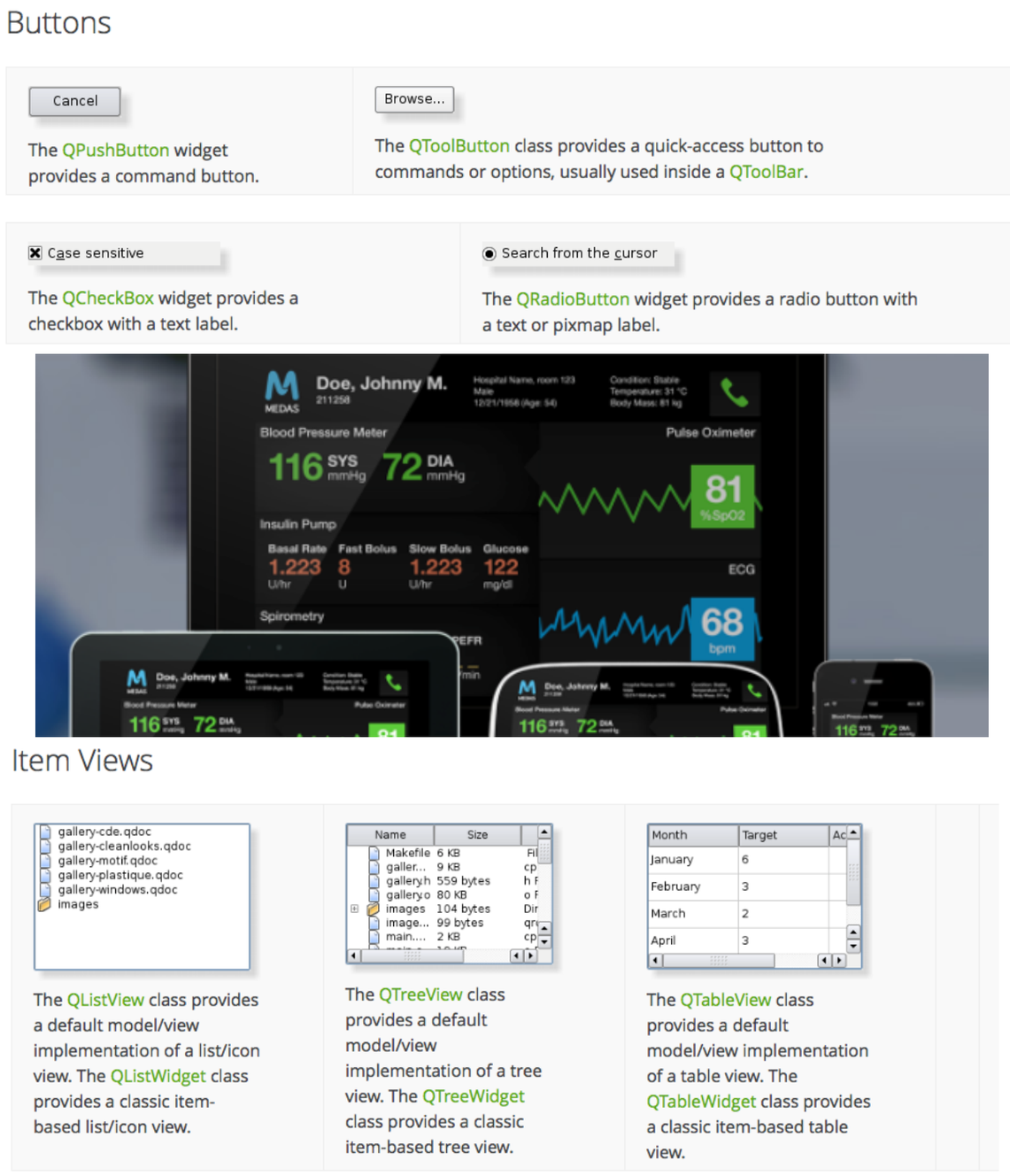
Python for Data Analysis and Visualization

- Python - excellent, mature language that offers extensive set of libraries and tools to developers
- Great for data analysis - offers tools like numpy, scipy, pandas, numba
- Excellent for data visualization with tools like Matplotlib for 2D/3D plotting
- Python bindings offered in many high quality C++ toolkits
- VTK - powerful C++ visualization framework offers Python bindings
- Several other UI frameworks offer compatibility with Python



Python for UI Development

- **Native toolkits like Cocoa on Mac can be used with Python - but this solution is not very portable even though offers best UI experience**
- **Multi-Platform UI Toolkits offer portability and extendibility to Python programmers - also offer great user experience by leveraging native widgets**
- **Several UI C++ Toolkits like WX and QT offer Python bindings**
- **QT - very rich and powerful UI Framework - offers Python binding through PySide and PyQt**
- **QT offers QML - declarative language - and plenty of widgets and some additional core functionality**
- **QT offers Slots/Signals for inter component communication, widgets and canvas are automatically double buffered**



Enaml - Python Native UI Language and Framework

- Enaml - declarative language that allows developers to quickly create dynamic user interfaces
- Enaml is pythonic - it extends Python language with its own syntax - but the result is compiled into python byte code
- Enaml currently supports QT widgets - probably one of the best multi-platform UI toolkits
- Enaml is influenced by QML - but it is pythonic (unlike QML) and can be debugged as python class - offers inheritance and other Python features
- Enaml offers additional features like dynamic scoping, powerful layout management, dynamic binding with Atom objects
- Enaml forces good programming practices - MVC - abstracting view from models, reusable components

```
enamldef Main(Window): main:
    attr counter = 0
    Container:
        constraints = [
            vbox(
                hbox(show_st, hide_st, spacer, ins_dyn, rem_dyn),
                nbook,
            ),
        ]
        PushButton: show_st:
            text = 'Show Static Pages'
            clicked ::
                static1.show()
                static2.show()
        PushButton: hide_st:
            text = 'Hide Static Pages'
            clicked ::
                static2.hide()
                static1.hide()
    PushButton: ins_dyn:
        text = 'Insert Dynamic Page'
        clicked ::
            title = 'Dynamic Page %s' % main.counter
            page = ContentPage(n=main.counter, title=title)
            dyn_pages.objects.insert(0, page)
            main.counter += 1
    PushButton: rem_dyn:
        text = 'Remove Dynamic Page'
        clicked ::
            if dyn_pages.objects:
                dyn_pages.objects.pop()
    Notebook: nbook:
        tab_style = 'document'
        Page: static1:
            title = 'Static Page1'
            Container:
                padding = 0
                Html:
                    source = '<h1><center>Static Page 1</center></h1>'
        Page: static2:
            title = 'Static Page2'
            Container:
                padding = 0
                Html:
                    source = '<h1><center>Static Page 2</center></h1>'
        Include: dyn_pages:
            pass
```

Enaml vs QML

- Some comparison of Enaml vs QML
- Both use same concept of declarative programming for user interfaces
- Both use Qt as backend UI framework and platform land leveraging rich set of Qt widgets
- Enaml is pythonic - QML uses JavaScript to do bindings
- QML provides more core Qt widgets, uses JSON format
- Enaml supports less core Qt widgets and it has it's own language and syntax but also provides some unique widgets
- Since Enaml is "pythonic" it integrates nicely with python and can be debugged as such and extended
- For Python programmers I would recommend Enaml over QML

```
//window containing the application
ApplicationWindow {

    //title of the application
    title: qsTr("Hello World")
    width: 640
    height: 480

    //menu containing two menu items
    menuBar: MenuBar {
        Menu {
            title: qsTr("File")
            MenuItem {
                text: qsTr("&Open")
                onTriggered: console.log("Open action triggered");
            }
            MenuItem {
                text: qsTr("Exit")
                onTriggered: Qt.quit();
            }
        }
    }

    //Content Area

    //a button in the middle of the content area
    Button {
        text: qsTr("Hello World")
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.verticalCenter: parent.verticalCenter
    }
}
```

Enaml UI offerings

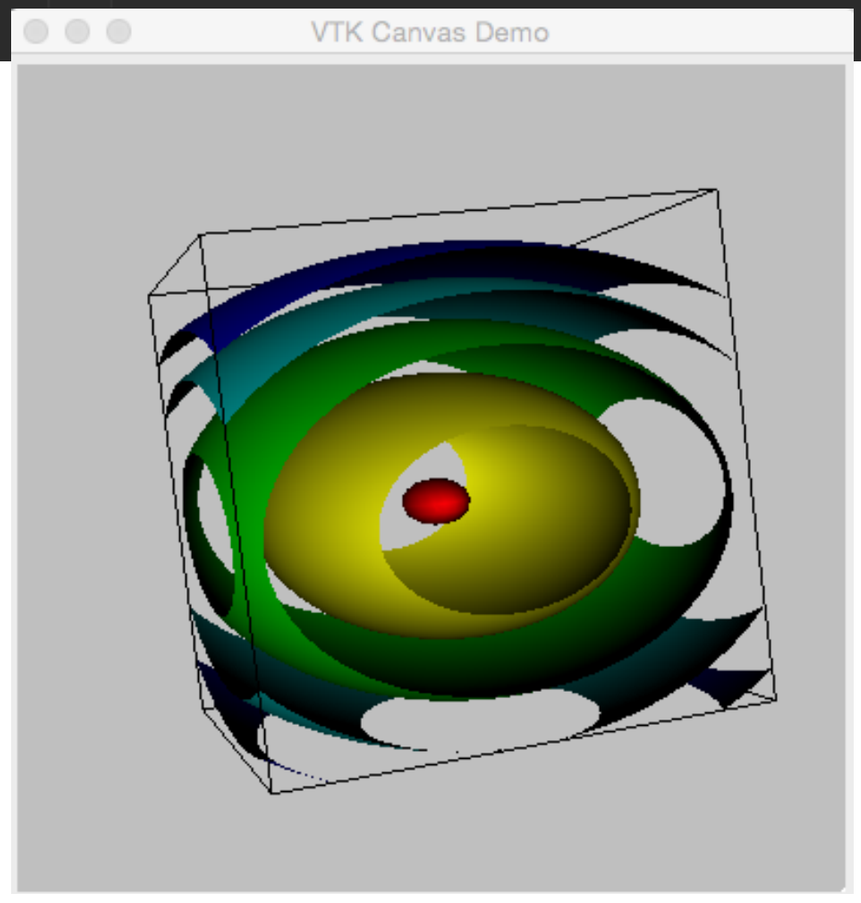
- Widgets such as Buttons, Labels, Combos, DateSelector, Popup Views
- Docking Layouts
- Dynamic and programmable user interface with Include, Looper and Condition
- Style Sheets and Templates
- Timer, TimedCalls, DeferredCalls

```
enamldef PlotDockItem( DockItem ): dock_item:

    attr figure_model = get_figure( title='Demo Prices', xlabel='Time', ylabel='Price')
    attr controller = ModelController( canvas=mpl.canvas, figure_model=figure_model )
    attr view_name

    name = view_name
    title = 'Price Plot Visualization'

    Container:
        padding = 5
        constraints = [ hbox( vbox( mpl, hbox( b, r, p ) ), ctrl ), b.width == r.width, p.width == b.width ]

        MPLCanvasContainer: mpl:
            mpl_figure << controller.figure_model.figure
            mpl_event_actions << [ ( 'key_press_event', on_key_press ), ]

        PushButton: b:
            text = 'Config'
            icon = load_icon('control-270.png')
            clicked ::
                ConfigPopup( self ).show()

        PushButton: r:
            text = 'Refresh'
            icon = load_icon( 'arrow-circle.png')
            clicked ::
                controller.plot_lines()

        PushButton: p:
            text = 'Add Plot'
            icon = load_icon( 'plus.png' )
            clicked ::
                addPlot( nonlocals.dock.dock_area, False )

        GroupBox: ctrl:
            title = 'Controls'
            padding = 5
            constraints = [ vbox( ba, o, spacer ) ]

            PushButton: ba:
                icon = load_icon( 'plus.png' )
                text = 'Add Data'
                clicked ::
                    controller.figure_model.add_data_set()
                    controller.figure_model.refresh_plot()

            Container: o:
                Include: inc:
                    objects << [ Controls( index=i, plot_controller=controller ) for i in xrange( controller.figure_model.current_data_len ) ]
```

VTK for immersive 3D and 4D data visualizations

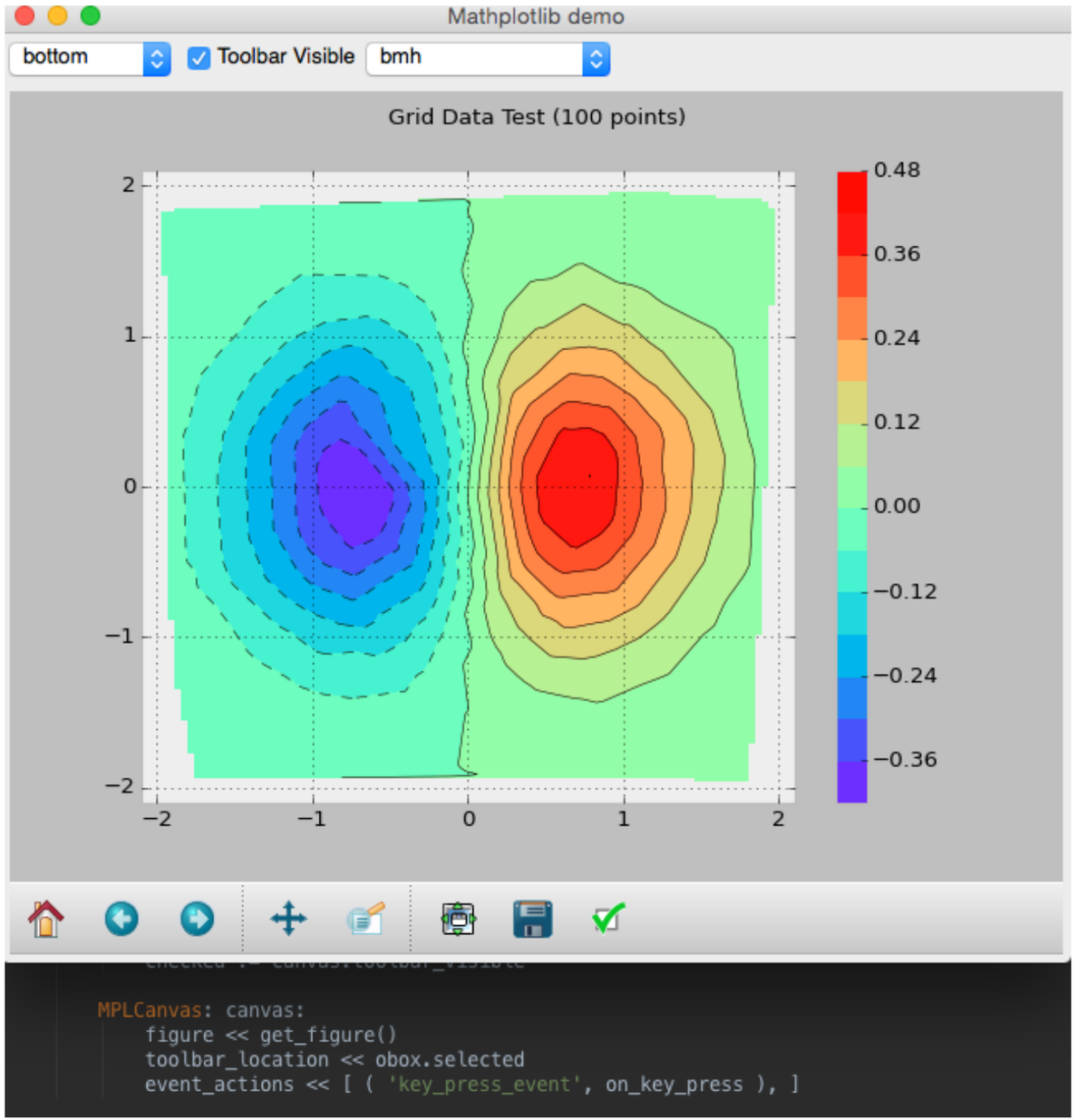
- VTK is a powerful OpenSource Data Visualization framework written in C++
- VTK offers Object Oriented approach to data visualization
- VTK uses OpenGL for rendering - speed really depends on the machine
- VTK can do 3D and 4D (volumetric) visualization and provides multitude of filters, components, functions - API could be quite overwhelming
- VTK provides many language bindings including Python - this makes use of VTK easier to understand and to use - specially in interactive environments
- Enaml provides VTKCanvas widget to integrate VTK in Enaml apps

```
enamldef Main(Window):
    attr custom_title = 'VTK Canvas Demo'
    title = custom_title

    Container:
        padding = 5
        VTKCanvas:
            renderer = create_renderer()
```



Matplotlib for 2D plotting and Realtime charting

- Matplotlib - OpenSource Python library and toolkit for high quality 2D plots - also for some 3D plots and contour plots, polar plots, multi charts, etc
- Highly customizable - outputs to variety of formats - png, pdf
- Supports various backends - including Qt
- Multitude of plot types - lines, bars, scatter, contours, etc
- Enaml provides MPLCanvas support for integrating Matplotlib with Enaml apps



Putting things together: Frameworks for interactive Data Visualization

- All tools discussed here are offered in Open Source Anaconda Python distribution 2.3.0 for Python 2.7 - update using conda if need it
- Matplotlib version 1.4.3
- Pandas version 0.17
- Numpy version 1.10.1
- Enaml version 0.9.8
- Atom version 0.3.9
- VTK - version 6.3
- QT - version 4.8

Enaml dynamic binding concepts explained

- **Enaml provides dynamic binding with use of Atom based classes - powerful feature that allows quick and easy development of interactive UI's**

- **Atom is C++/Python framework that provides memory saving, speed, and dynamic bi-directional bindings between models and UI components**

- **Atom provides state notifications - when member is notified of change - actions can be performed**

```python
from atom.api import Atom, Unicode, Range, Bool, observe


class Person(Atom):
    """ A simple class representing a person object.

    """
    last_name = Unicode()

    first_name = Unicode()

    age = Range(low=0)

    debug = Bool(False)

    @observe('age')
    def debug_print(self, change):
        """ Prints out a debug message whenever the person'

        """
        if self.debug:
            templ = "{first} {last} is {age} years old."
            s = templ.format(
                first=self.first_name, last=self.last_name,
            )
            print(s)

    def _default_first_name(self):
        return 'John'


john = Person(last_name='Doe', age=42)
john.debug = True
john.age = 43  # prints message
john.age = 'forty three'   # raises TypeError
```

Enaml is Python - create new components as you would new classes in Python

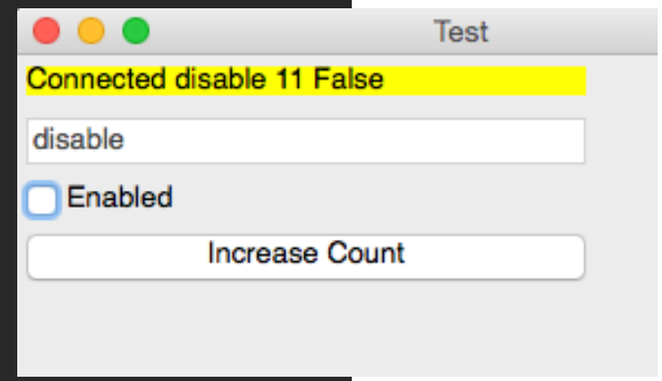
- Enaml offers good range of widgets (Qt based)
- Sometimes you need to create new widgets or extend existing one by adding new functionality
- Creating new Slider widget by encapsulating and extending existing one
- Expose new attributes to the outside world

```
from enaml.layout.api import hbox, spacer, vbox, align
from enaml.widgets.api import Container, Border, Label, Slider


enamldef SliderControl( Container ):
    ''' custom slider control '''

    attr info_label      = 'Test'
    attr min_value       = 0
    attr max_value       = 100
    attr slider_value    = 0
    attr step_value      = 1
    attr page_step_value = 10
    attr interval_value  = 5
    attr tick_postion    = 'no_ticks'

    border      = Border( line_style='plain', line_width=1 )
    padding     = ( 5, 5, 5, 5 )
    foreground  = '#cdcdcd'
    constraints = [
        vbox( l, hbox( mi, spacer, v, spacer, mx ), sl ),
        align( 'left', mi, sl ),
        align( 'right', mx, sl ),
        align( 'h_center', v, sl ),
    ]

    Slider: sl:
        minimum       = min_value
        maximum       = max_value
        single_step   = step_value
        page_step     = page_step_value
        tick_interval = interval_value
        tick_position = tick_postion
        hug_width     = 'weak'
        value         := slider_value

    Label: l:
        text = info_label
        foreground  = '#000000'

    Label: v:
        text << '%d' % sl.value
        foreground  = '#000000'

    Label: mi:
        text = '%d' % min_value
        foreground  = '#000000'

    Label: mx:
        text = '%d' % max_value
        foreground  = '#000000'
```

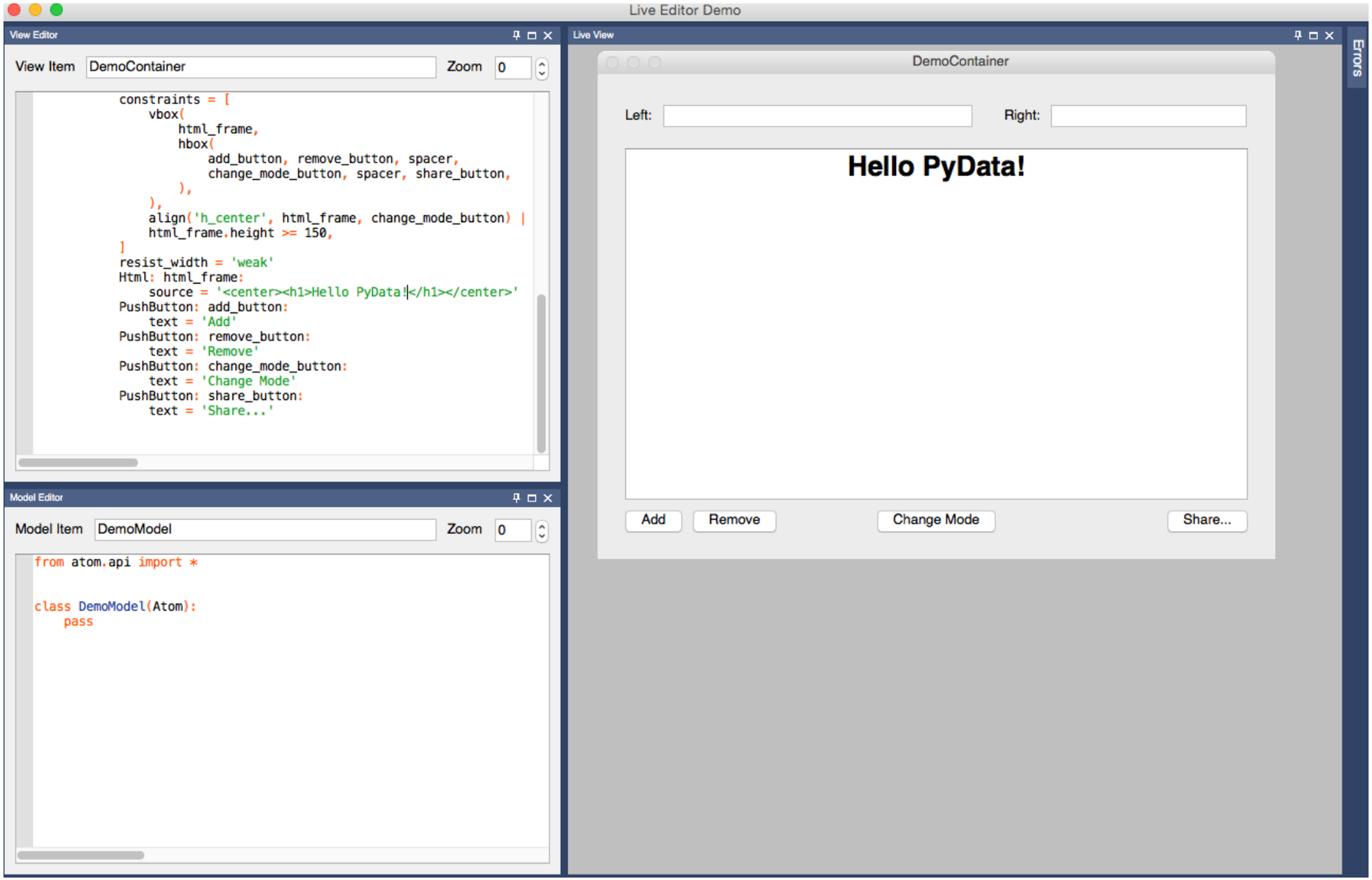
How Enaml binds with your model using Atom

- Use Enaml to create application user interface by declaring components and layouts
- Use Atom based controller as a conduit between your application interface and your visualization model
- Enaml uses dynamic subscription with the controller that upon interaction can modify your model and vice versa
- Changes in Model can be reflected in UI interface
- All required is to wire things properly between components and model
- Use following operators to control component subscription:
  - = simple assignment
  - >> changes from UI flow to the Model
  - << changes from Model update UI
  - := bi-directional subscription
  - :: perform action

```
from enaml.widgets.api import Window, Container, Label, Field, CheckBox, PushButton
from enaml.layout.api import vbox, hbox, spacer, align
from atom.api import Atom, Str, Int, Bool

class Model(Atom):

    status = Str('Connected')
    message = Str('')
    enabled = Bool(default=False)
    counter = Int(default=0)
    bg_color = Str('white')

    def _observe_message(self, change):
        if change:
            value = change.get('value')
            if value == 'enable':
                self.enabled = True
            if value == 'disable':
                self.enabled = False

    def _observe_counter(self, change):
        if change:
            value = change.get('value')
            if value > 10:
                self.bg_color = 'yellow'
            if value > 20:
                self.bg_color = 'red'

enamldef Main(Window):

    title = 'Test'
    minimum_size = ( 400, 400 )

    attr model = Model()

    Container:
        padding = 5
        constraints = [ vbox( l, f, c, b ), align( 'left', l, f, c, b ) ]

        Label: l:
            background << model.bg_color
            text << ( model.status + ' ' + model.message + ' ' +
                      str( model.counter ) + ' ' +
                      str( model.enabled ) )

        Field: f:
            text >> model.message
            enabled << model.enabled

        CheckBox: c:
            constraints = [ width == 250 ]
            text = 'Enabled'
            checked := model.enabled

        PushButton: b:
            text = 'Increase Count'
            clicked ::
                model.counter += 1
```



Use Enaml interactive editor (REPL) to quickly mock your layouts - requires QScintilla package for Qt



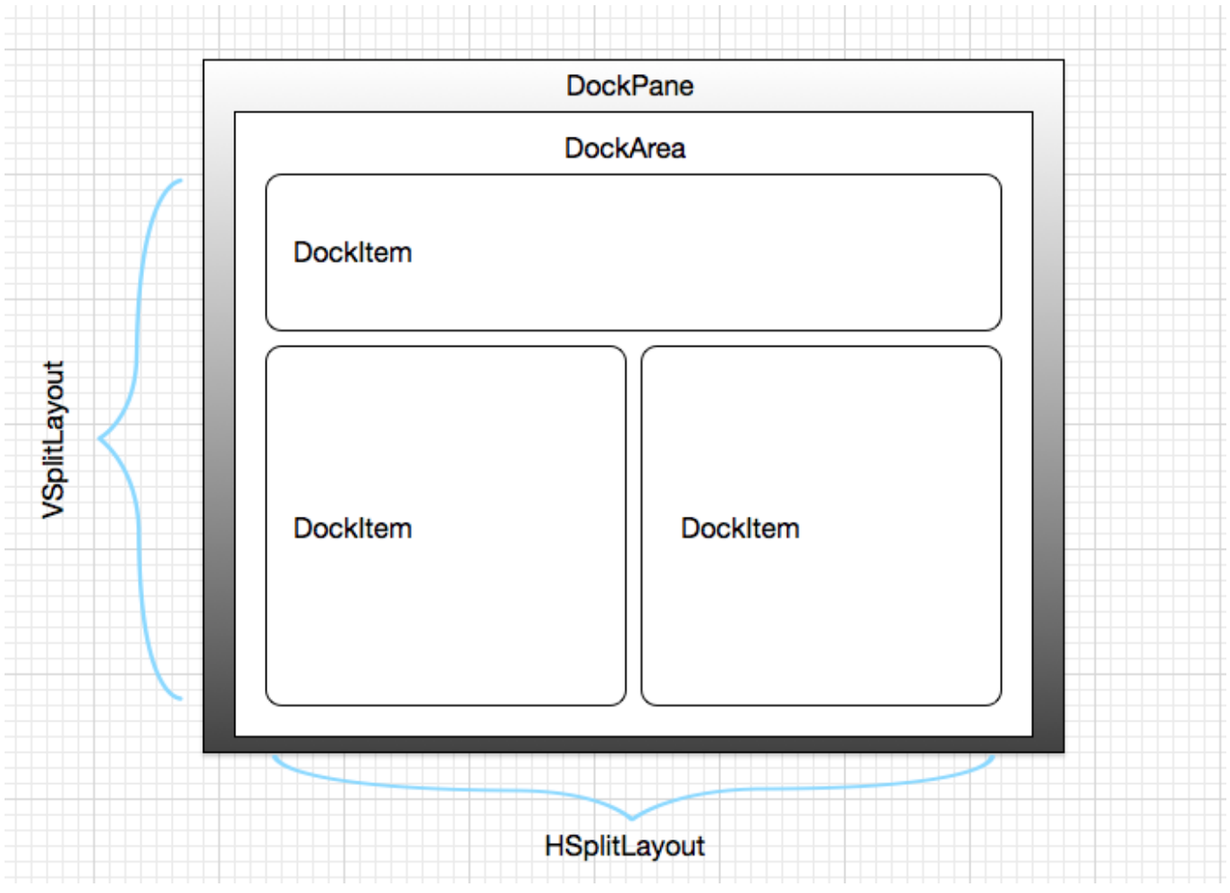
Use Enaml widgets and layouts to design visual look of your application

- **Enaml offers variety of widgets and layout constraints to simplify user interface design and development**

- **Useful widgets such as Forms, PopupView, DockArea, DockItems**

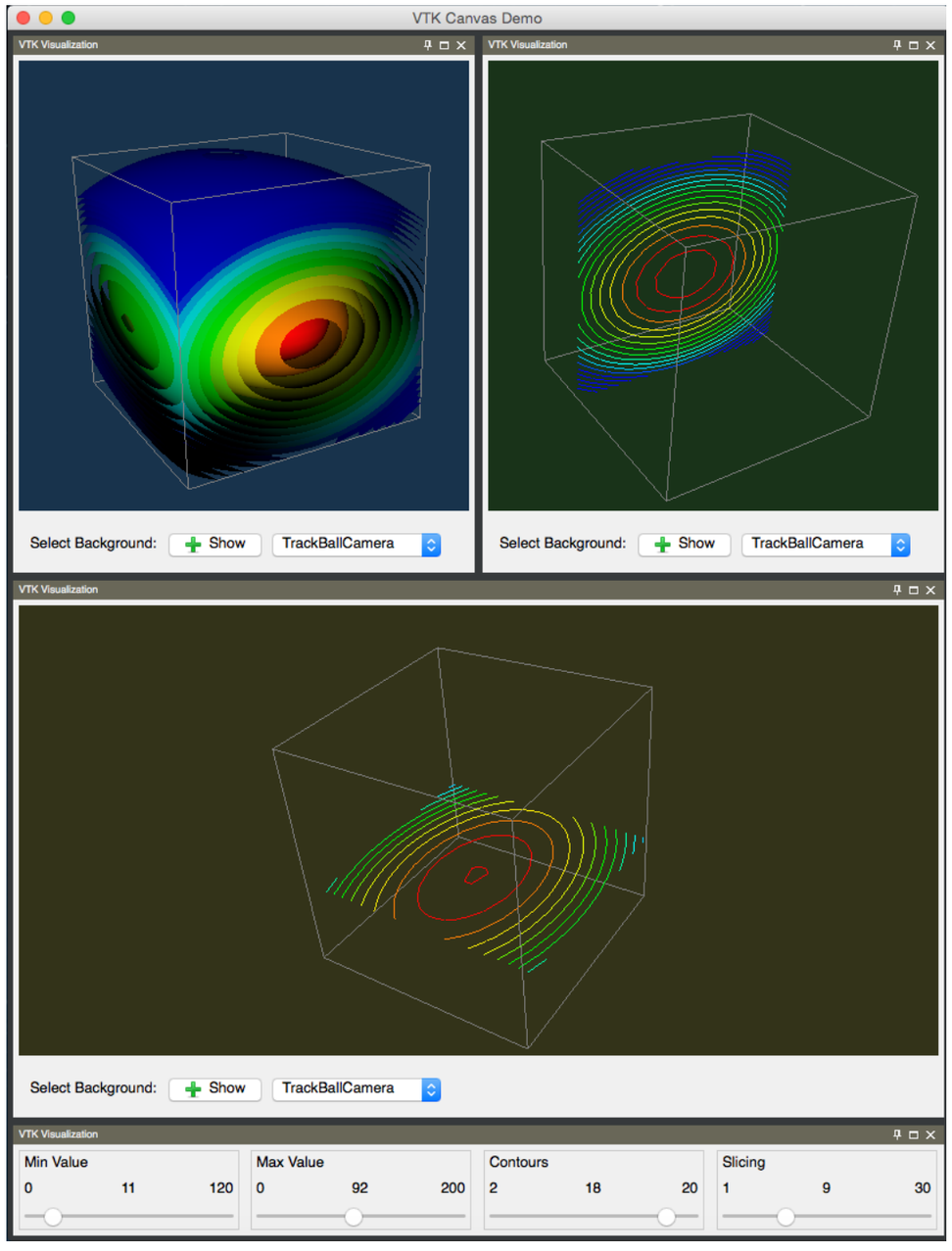
- **Use style sheet to fine tune look and feel**

- **Use Include and Conditional to dynamically control your application**



Typical layout of the visualization application with multiple views

- Multiple docking windows show different aspect of the visualization
- Controls with sliders allows different selection of the underlying data
- Probing and slicing data interactively allows better understanding of data
- Popup Views offer quick access to model properties



Bootstrap Enaml file in Python with QT Application

- Enaml has support in some editors like Sublime Text
- Some editors like PyCharm can be easily extended to support Enaml syntax
- You can run Enaml file from command line: enaml-run <your-file.enaml>
- Bootstrap Enaml file in the code
- With bootstrapping you can pass your arguments into Enaml file and you can also run in debug mode

```
import enaml
from enaml.qt.qt_application import QtApplication


def run_demo():
    """
    bootstrap enaml in python
    """
    with enaml.imports():
        from vtk_demo_ui import Main

    app = QtApplication()

    view = Main(custom_title='VTK Canvas Demo')
    view.show()

    # Start the application event loop
    app.start()


run_demo()
```

Quick intro to VTK visualization pipeline flow

- VTK API offers hundreds of components, functions, filters, scalers, transitions, mappers, etc - it can be quite overwhelming
- VTK has been around for many years - since 1995 as Object Oriented way to create powerful visualization
- But in practice most of the creation of the visualization comes to the simple pipeline flow
- Map data set into specific data object mapper
- Apply filter or other functions on the mapper
- Apply scaling factors, color map, set data boundaries
- Feed mapper into Actor, setup camera and lights and feed all this into renderer object and render Window

```python
import vtk


def create_renderer():
    """
    vtk renderer with a sample scene
    """
    quadric = vtk.vtkQuadric()
    quadric.SetCoefficients(.5, 1, .2, 0, .1, 0, 0, .2, 0, 0)

    sample = vtk.vtkSampleFunction()
    sample.SetSampleDimensions(50, 50, 50)
    sample.SetImplicitFunction(quadric)

    contours = vtk.vtkContourFilter()
    contours.SetInputConnection(sample.GetOutputPort())
    contours.GenerateValues(5, 0.0, 1.2)

    contour_mapper = vtk.vtkPolyDataMapper()
    contour_mapper.SetInputConnection(contours.GetOutputPort())
    contour_mapper.SetScalarRange(0.0, 1.2)

    contour_actor = vtk.vtkActor()
    contour_actor.SetMapper(contour_mapper)

    outline = vtk.vtkOutlineFilter()
    outline.SetInputConnection(sample.GetOutputPort())

    outline_mapper = vtk.vtkPolyDataMapper()
    outline_mapper.SetInputConnection(outline.GetOutputPort())

    outline_actor = vtk.vtkActor()
    outline_actor.SetMapper(outline_mapper)
    outline_actor.GetProperty().SetColor(0, 0, 0)

    renderer = vtk.vtkRenderer()
    renderer.AddActor(contour_actor)
    renderer.AddActor(outline_actor)
    renderer.SetBackground(.75, .75, .75)

    return renderer
```

VTK Class Index

Class Index

2 | 3 | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | _

2

vtk2DHistogramItem

3

vtk3DSImporter
vtk3DWidget

A

vtkABI
vtkAbstractArray
vtkAbstractCellLocator
vtkAbstractContextBufferId
vtkAbstractContextItem
vtkAbstractElectronicData
vtkAbstractGridConnectivity
vtkAbstractImageInterpolator
vtkAbstractInteractionDevice
vtkAbstractInterpolatedVelocityField
vtkAbstractMapper
vtkAbstractMapper3D
vtkAbstractParticleWriter
vtkAbstractPicker
vtkAbstractPointLocator
vtkAbstractPolygonalHandleRepresentation3D
vtkAbstractPropPicker
vtkAbstractRenderDevice
vtkAbstractTransform
vtkAbstractVolumeMapper
vtkAbstractWidget
vtkAcceleratorsDaxObjectFactory
ActionFunction
vtkActor
vtkActor2D
vtkActor2DCollection
vtkActorCollection
vtkAddMembershipArray
vtkADIOSDirTree
vtkADIOSReader
vtkADIOSWriter
vtkAdjacencyMatrixToEdgeTable
vtkAdjacentVertexIterator
vtkAffineRepresentation
vtkAffineRepresentation2D
vtkAffineWidget
vtkAlgorithm
vtkAlgorithmOutput
vtkAmoebaMinimizer
vtkAMRBaseParticlesReader
vtkAMRBaseReader
vtkAMRBox
vtkAMRCutPlane
vtkAMRDataInternals
vtkAMRDataSetCache
vtkAMREnzoParticlesReader
vtkAMREnzoReader
vtkAMREnzoReaderInternal
vtkAMRFlashParticlesReader
vtkAMRFlashReader
vtkAMRFlashReaderInternal
vtkAMRGaussianPulseSource
vtkAMRInformation
vtkAMRInterpolatedVelocityField
vtkAMRResampleFilter
vtkAMRSliceFilter
vtkAMRToMultiBlockFilter
vtkAMRUtilities
vtkAMRVolumeMapper
vtkAndroidOutputWindow
vtkAndroidRenderWindowInteractor
vtkAngleRepresentation
vtkAngleRepresentation2D
vtkAngleRepresentation3D
vtkAngleWidget
vtkAnimationCue
vtkAnimationCue::AnimationCueInfo
vtkAnimationScene
vtkAnnotatedCubeActor
vtkAnnotation
vtkAnnotationLayers
vtkAnnotationLayersAlgorithm
vtkAnnotationLink
vtkAppendCompositeDataLeaves
vtkAppendFilter
vtkAppendPoints
vtkAppendPolyData
vtkAppendSelection
vtkApplyColors
vtkApplyIcons
vtkApproximatingSubdivisionFilter
vtkArcParallelEdgeStrategy
vtkArcPlotter
vtkHierarchicalPolyDataMapper
vtkHighestDensityRegionsStatistics
vtkHomogeneousTransform
vtkHoverWidget
vtkHull
vtkHyperOctree
vtkHyperOctreeAlgorithm
vtkHyperOctreeClipCutPointsGrabber
vtkHyperOctreeContourFilter
vtkHyperOctreeCursor
vtkHyperOctreeCutter
vtkHyperOctreeDepth
vtkHyperOctreeDualGridContourFilter
vtkHyperOctreeFractalSource
vtkHyperOctreeLightWeightCursor
vtkHyperOctreeLimiter
vtkHyperOctreePointsGrabber
vtkHyperOctreeSampleFunction
vtkHyperOctreeSurfaceFilter
vtkHyperOctreeToUniformGridFilter
vtkHyperStreamline
vtkHyperTree
vtkHyperTreeCursor
vtkHyperTreeGrid
vtkHyperTreeGridAlgorithm
vtkHyperTreeGridAxisCut
vtkHyperTreeGridGeometry
vtkHyperTreeGridSource
vtkHyperTreeGrid::vtkHyperTreeGridSuperCursor
vtkHyperTreeGridToUnstructuredGrid
vtkHyperTreeGrid::vtkHyperTreeIterator
vtkHyperTreeGrid::vtkHyperTreeSimpleCursor

I

vtkIcicleView
vtkIconGlyphFilter
vtkIdentityTransform
vtkIdFilter
vtkIdList
vtkIdListCollection
vtkIdTypeArray
vtkImage2DIslandPixel
vtkImageAccumulate
vtkImageActor
vtkImageActorPointPlacer
vtkImageAlgorithm
vtkImageAnisotropicDiffusion2D
vtkImageAnisotropicDiffusion3D
vtkImageAppend
vtkImageAppendComponents
vtkImageBlend
vtkImageBSplineCoefficients
vtkImageBSplineInternals
vtkImageBSplineInterpolator
vtkImageButterworthHighPass
vtkImageButterworthLowPass
vtkImageCacheFilter
vtkImageCanvasSource2D
vtkImageCast
vtkImageChangeInformation
vtkImageCheckerboard
vtkImageCityBlockDistance
vtkImageClip
vtkImageComplex
vtkImageConnector
vtkImageConnectorSeed
vtkImageConstantPad
vtkImageContinuousDilate3D
vtkImageContinuousErode3D
vtkImageConvolve
vtkImageCorrelation
vtkImageCroppingRegionsWidget
vtkImageCursor3D
vtkImageData
vtkImageDataGeometryFilter
vtkImageDataLIC2D
vtkImageDataStreamer
vtkImageDataToPointSet
vtkImageDataToUniformGrid
vtkImageDecomposeFilter
vtkImageDifference
vtkImageDilateErode3D
vtkImageDivergence
vtkImageDotProduct
vtkImageEllipsoidSource
vtkImageEuclideanDistance
vtkImageEuclideanToPolar
vtkImageExport
vtkImageExtractComponents
vtkImageFFT
vtkImageFlip
vtkImageFourierCenter
vtkProgressObserver
vtkProjectedAAHexahed
vtkProjectedTerrainPat
vtkProjectedTetrahedra
vtkProjectedTexture
vtkProjectSphereFilter
vtkProp
vtkProp3D
vtkProp3DAxisFollower
vtkProp3DButtonRepre
vtkProp3DCollection
vtkProp3DFollower
vtkPropAssembly
vtkPropCollection
vtkProperty
vtkProperty2D
property_map< vtkDire
property_map< vtkDire
property_map< vtkDire
property_map< vtkDire
property_map< vtkGra
property_map< vtkGra
property_map< vtkGra
property_map< vtkGra
property_map< vtkGra
property_map< vtkUnd
property_map< vtkUnd
property_map< vtkUnd
property_map< vtkUnd
property_traits< vtkAb
property_traits< vtkDa
property_traits< vtkEd
property_traits< vtkGr
property_traits< vtkGr
property_traits< vtkVe
property_traits< vtkVe
property_traits< vtkVe
vtkPropPicker
vtkProStarReader
vtkProteinRibbonFilter
vtkPruneTreeFilter
vtkPSLACReader
vtkPSphereSource
vtkPStreaklineFilter
vtkPStreamTracer
vtkPStructuredGridCon
vtkPStructuredGridGho
vtkPSurfaceLICCompos
vtkPSurfaceLICPainter
vtkPTemporalStreamTr
vtkPUniformGridGhostI
vtkPUnstructuredGridC
vtkPUnstructuredGridG
vtkPVWebGLExporter
vtkPWindBladeReader
vtkPYoungsMaterialInte
vtkPyramid
vtkPythonAlgorithm
vtkPythonArgs
vtkPythonCommand
vtkPythonOverload
vtkPythonUtil

Q

vtkQImageToImageSou
vtkQtAbstractModelAda
vtkQtAnnotationLayers
vtkQtAnnotationView
vtkQtConnection
vtkQtDebugLeaksMode
vtkQtDebugLeaksView
vtkQtInitialization
vtkQtLabelRenderStrate
vtkQtListView
vtkQtRecordView
vtkQtRichTextView
vtkQtSQLDatabase
vtkQtSQLQuery
vtkQtStringToImage
vtkQtTableModelAdapt
vtkQtTableRepresentat
vtkQtTableView
vtkQtTimePointUtility
vtkQtTreeModelAdapte
vtkQtTreeRingLabelMa
vtkQtTreeView
vtkQtView
vtkQuad
vtkQuadraticEdge
vtkQuadraticHexahedro
vtkQuadraticLinearQua
vtkAppendSelection
vtkApplyColors
vtkApplyIcons
vtkApproximatingSubdivisionFilter
vtkArcParallelEdgeStrategy
vtkArcPlotter
vtkArcSource
vtkAreaContourSpectrumFilter
vtkAreaLayout
vtkAreaLayoutStrategy
vtkAreaPicker
vtkArray
vtkArrayCalculator
vtkArrayCoordinates
vtkArrayData
vtkArrayDataAlgorithm
vtkArrayDataReader
vtkArrayDataWriter
vtkArrayExtents
vtkArrayExtentsList
vtkExodusIIReaderPrivate::ArrayInfoType
vtkArrayInterpolate
vtkArrayIterator
vtkArrayIteratorIncludes
vtkArrayIteratorTemplate
vtkArrayNorm
vtkArrayPrint
vtkArrayRange
vtkArrayReader
vtkArraySort
vtkArrayToTable
vtkArrayWeights
vtkArrayWriter
vtkArrowSource
vtkASCIITextCodec
vtkAssembly
vtkExodusIIReaderPrivate::AssemblyInfoType
vtkAssemblyNode
vtkAssemblyPath
vtkAssemblyPaths
vtkStatisticsAlgorithm::AssessFunctor
vtkAssignAttribute
vtkAssignCoordinates
vtkAssignCoordinatesLayoutStrategy
vtkAtom
vtkAtomicInt
vtkAtomicIntImpl (detail)
vtkAtomicIntImpl< vtkTypeInt32 > (detail)
vtkAtomicIntImpl< vtkTypeInt64 > (detail)
vtkAttributeClustering2DLayoutStrategy
vtkAttributeDataToFieldDataFilter
vtkAttributesErrorMetric
vtkAutoCorrelativeStatistics
vtkAVIWriter
vtkAVSucdReader
vtkAxes
vtkAxesActor
vtkAxesTransformRepresentation
vtkAxesTransformWidget
vtkAxis
vtkAxisActor
vtkAxisActor2D
vtkAxisExtended
vtkAxisFollower

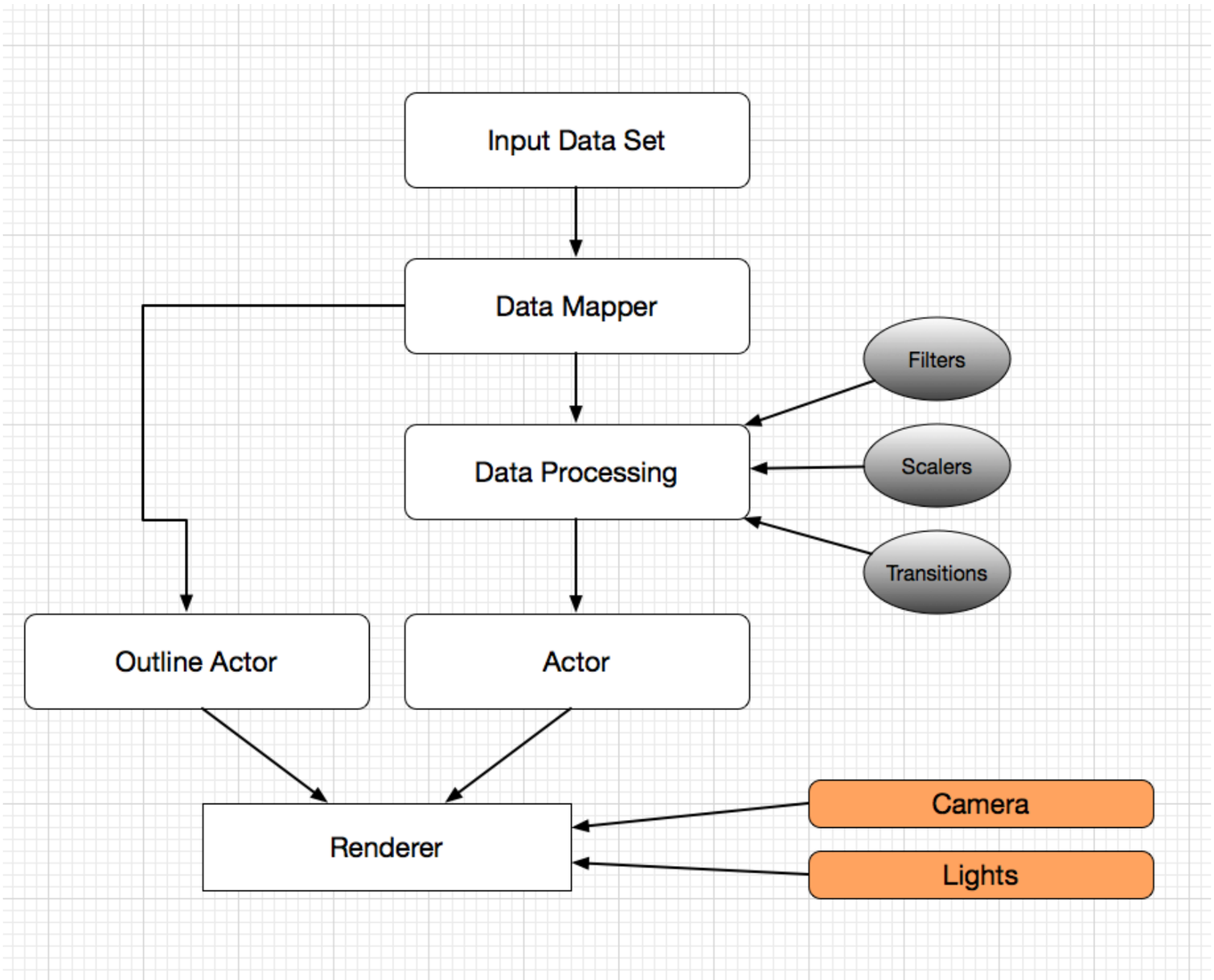
B

vtkBackgroundColorMonitor
vtkBalloonRepresentation
vtkBalloonWidget
vtkBandedPolyDataContourFilter
vtkBarChartActor
vtkBase64InputStream
vtkBase64OutputStream
vtkBase64Utilities
vtkFieldData::BasicIterator
vtkBezierContourLineInterpolator
vtkBiDimensionalRepresentation
vtkBiDimensionalRepresentation2D
vtkBiDimensionalWidget
vtkBiomTableReader
vtkBiQuadraticQuad
vtkBiQuadraticQuadraticHexahedron
vtkBiQuadraticQuadraticWedge
vtkBiQuadraticTriangle
vtkBitArray
vtkBitArrayIterator
vtkBivariateLinearTableThreshold
vtkBlankStructuredGrid
vtkBlankStructuredGridWithImage
vtkExodusIIWriter::Block
vtkAMRDataInternals::Block
vtkBlockDistribution
vtkBlockIdScalars
vtkExodusIIReaderPrivate::BlockInfoType
vtkBlockItem
vtkImageEuclideanToPolar
vtkImageExport
vtkImageExtractComponents
vtkImageFFT
vtkImageFlip
vtkImageFourierCenter
vtkImageFourierFilter
vtkImageGaussianSmooth
vtkImageGaussianSource
vtkImageGradient
vtkImageGradientMagnitude
vtkImageGridSource
vtkImageHistogram
vtkImageHistogramStatistics
vtkImageHSIToRGB
vtkImageHSVToRGB
vtkImageHybridMedian2D
vtkImageIdealHighPass
vtkImageIdealLowPass
vtkImageImport
vtkImageImportExecutive
vtkImageInPlaceFilter
vtkImageInterpolator
vtkImageIslandRemoval2D
vtkImageItem
vtkImageIterateFilter
vtkImageIterator
vtkImageLaplacian
vtkImageLogarithmicScale
vtkImageLogic
vtkImageLuminance
vtkImageMagnify
vtkImageMagnitude
vtkImageMandelbrotSource
vtkImageMapper
vtkImageMapper3D
vtkImageMapToColors
vtkImageMapToRGBA
vtkImageMapToWindowLevelColors
vtkImageMarchingCubes
vtkImageMask
vtkImageMaskBits
vtkImageMathematics
vtkImageMedian3D
vtkImageMirrorPad
vtkImageNoiseSource
vtkImageNonMaximumSuppression
vtkImageNormalize
vtkImageOpenClose3D
vtkImageOrthoPlanes
vtkImagePadFilter
vtkImagePermute
vtkImagePlaneWidget
vtkImageProcessingPass
vtkImageProgressIterator
vtkImageProperty
vtkImageQuantizeRGBToIndex
vtkImageRange3D
vtkImageReader
vtkImageReader2
vtkImageReader2Collection
vtkImageReader2Factory
vtkImageRectilinearWipe
vtkImageRenderManager
vtkImageResample
vtkImageResize
vtkImageReslice
vtkImageResliceMapper
vtkImageResliceToColors
vtkImageRFFT
vtkImageRGBToHSI
vtkImageRGBToHSV
vtkImageSeedConnectivity
vtkImageSeparableConvolution
vtkImageShiftScale
vtkImageShrink3D
vtkImageSincInterpolator
vtkImageSinusoidSource
vtkImageSkeleton2D
vtkImageSlab
vtkImageSlabReslice
vtkImageSlice
vtkImageSliceCollection
vtkImageSliceMapper
vtkImageSobel2D
vtkImageSobel3D
vtkImageSpatialAlgorithm
vtkImageStack
vtkImageStencil
vtkImageStencilAlgorithm
vtkImageStencilData
vtkImageStencilIterator
vtkImageStencilRaster
vtkImageStencilSource
vtkImageStencilToImage
vtkQtTreeView
vtkQtView
vtkQuad
vtkQuadraticEdge
vtkQuadraticHexahedro
vtkQuadraticLinearQuad
vtkQuadraticLinearWedg
vtkQuadraticPolygon
vtkQuadraticPyramid
vtkQuadraticQuad
vtkQuadraticTetra
vtkQuadraticTriangle
vtkQuadraticWedge
vtkQuadraturePointInter
vtkQuadraturePointsGer
vtkQuadratureSchemeDe
vtkQuadratureSchemeDi
vtkQuadric
vtkQuadricClustering
vtkQuadricDecimation
vtkQuadricLODActor
vtkQuadRotationalExtru
vtkQuantizePolyDataPoi
vtkQuaternion
vtkQuaternionInterpolat

R

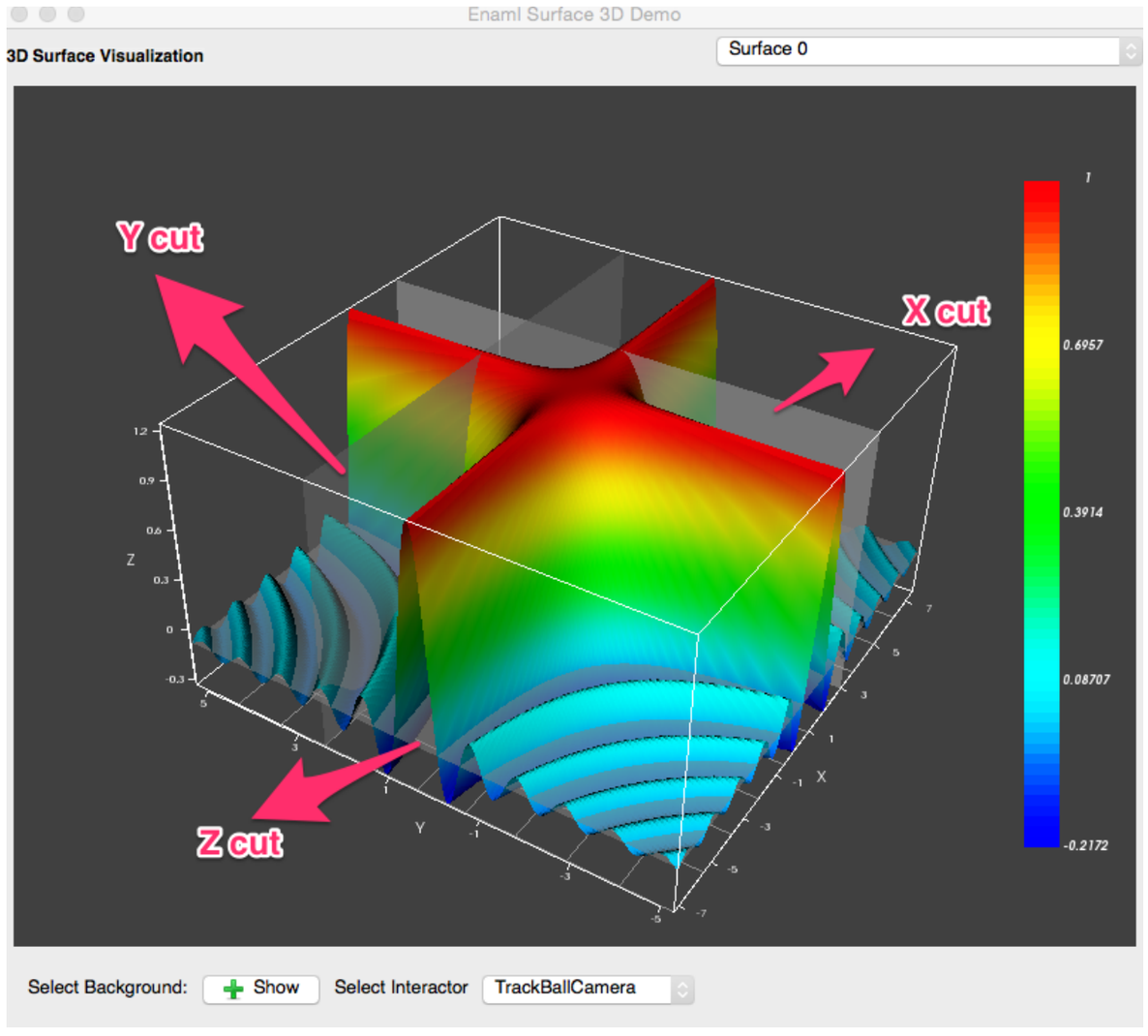
vtkRAdapter
vtkRandomAttributeGen
vtkRandomGraphSource
vtkRandomLayoutStrate
vtkRandomSequence
vtkSynchronizedRendere
vtkRayCastImageDisplay
vtkRayCastRayInfo
vtkRayCastStructures
vtkRCalculatorFilter
vtkRearrangeFields
vtkRect
vtkRectangularButtonSo
vtkRectd
vtkRectf
vtkRecti
vtkRectilinearGrid
vtkRectilinearGridAlgori
vtkRectilinearGridClip
vtkRectilinearGridGeome
vtkRectilinearGridOutlin
vtkRectilinearGridPartiti
vtkRectilinearGridReade
vtkRectilinearGridToPoi
vtkRectilinearGridToTet
vtkRectilinearGridWriter
vtkRectilinearSynchroni
vtkRectilinearWipeRepre
vtkRectilinearWipeWidge
vtkRecursiveDividingCul
vtkRecursiveSphereDire
vtkReduceTable
vtkReebGraph
vtkReebGraphSimplificat
vtkReebGraphSimplificat
vtkReebGraphSurfaceSk
vtkReebGraphToJoinSpli
vtkReebGraphVolumeSk
vtkReferenceCount
ReferenceCountModel
vtkReflectionFilter
vtkRegressionTester
vtkRegularPolygonSourc
vtkPParticleTracerBase::I
vtkRemoveHiddenData
vtkRemoveIsolatedVertic
vtkGenericCompositePol
vtkCompositePainter::Re
vtkRenderbuffer
vtkRenderedAreaPicker
vtkRenderedGraphRepre
vtkRenderedHierarchyRe
vtkRenderedRepresentat
vtkRenderedSurfaceRep
vtkRenderedTreeAreaRe
vtkRenderer
vtkRendererCollection
vtkRendererDelegate
vtkSynchronizedRendere
vtkParallelRenderManag
vtkRendererSource
vtkRenderingContextOp
vtkRenderingFreeTypeO
vtkRenderingFreeTypeO
vtkRenderingOpenGLOb
vtkRenderingVolumeOp

VTK simplified visualization pipeline diagram

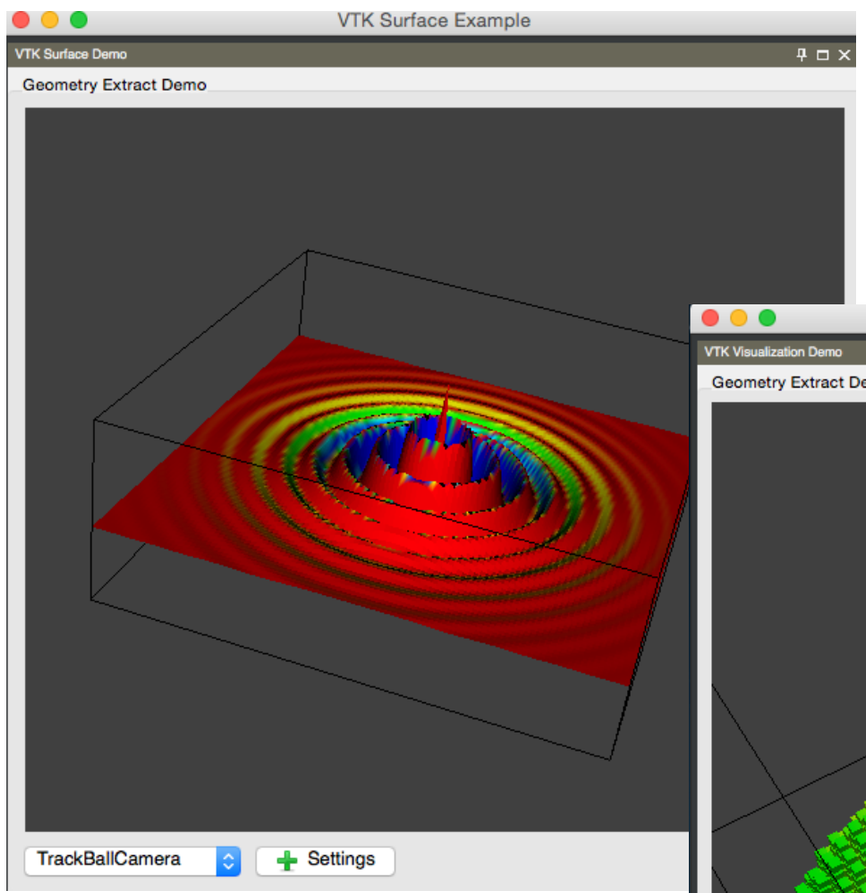


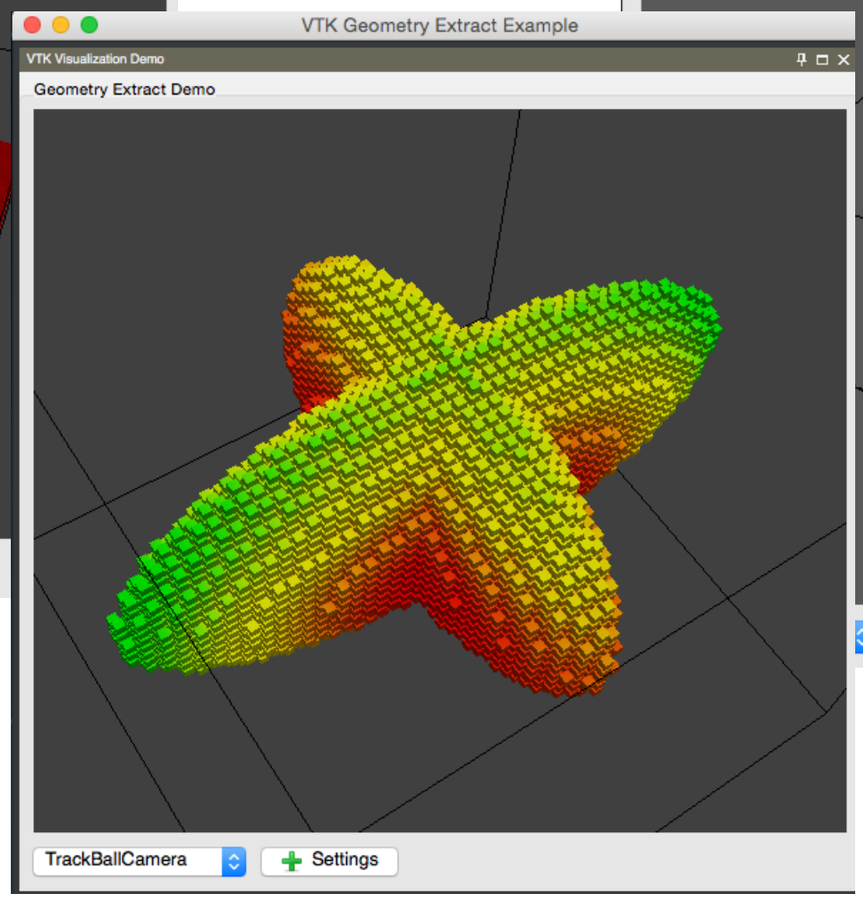
Dynamically Control your VTK visualization model with Enaml

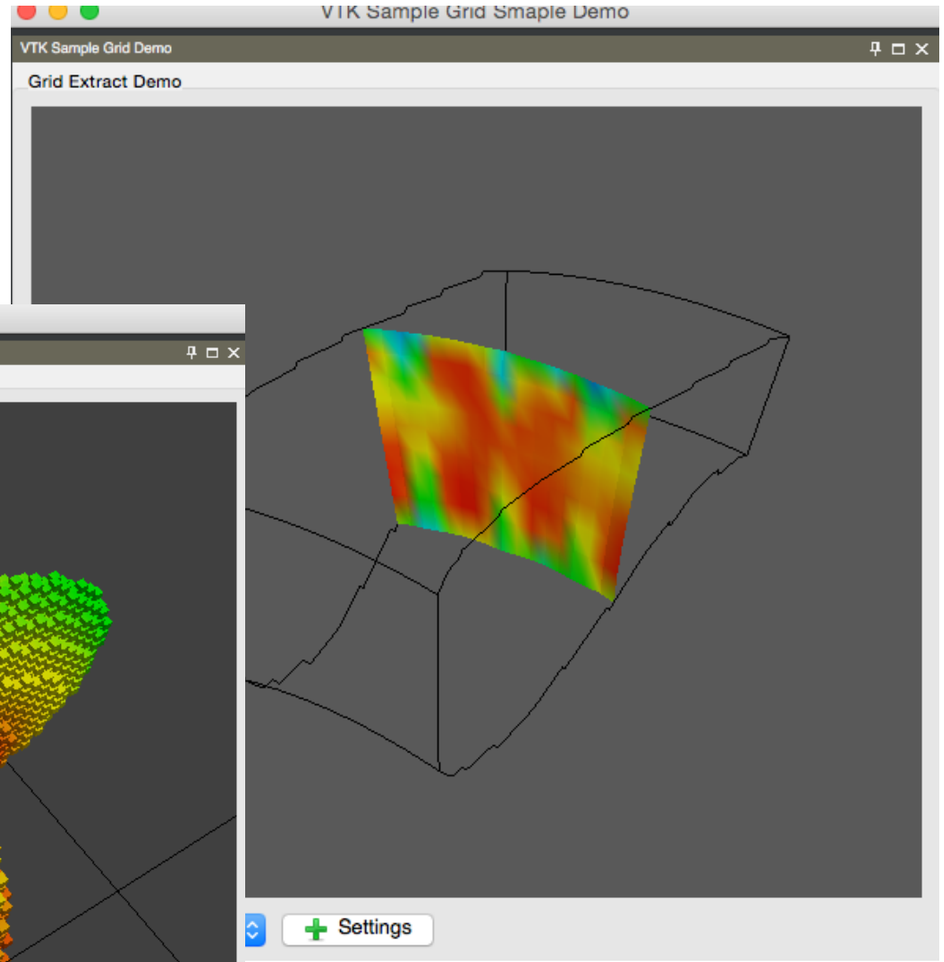
- Apply changes at any point in the visualization pipeline
- Change scaling and resolution
- Apply different colors and mappings
- Slicing and projecting
- Contours controls



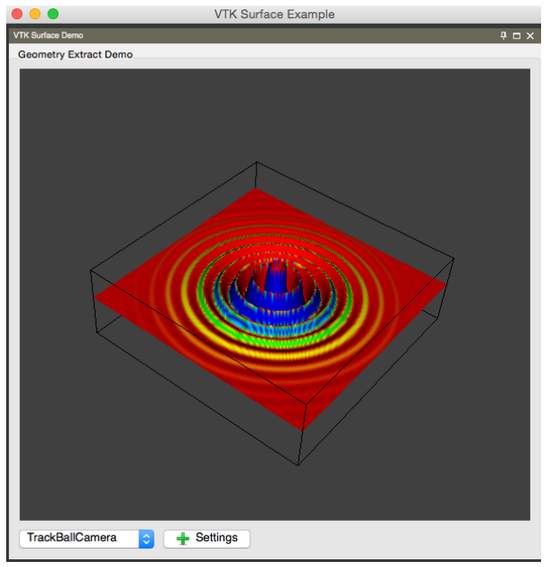
VTK Enaml Interaction in Examples and Code Samples



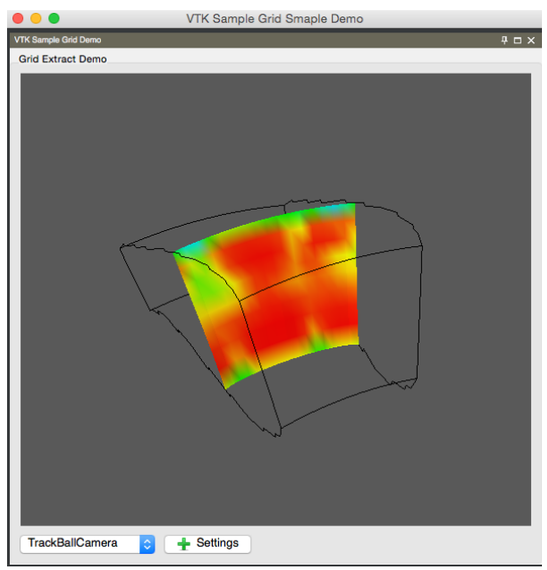




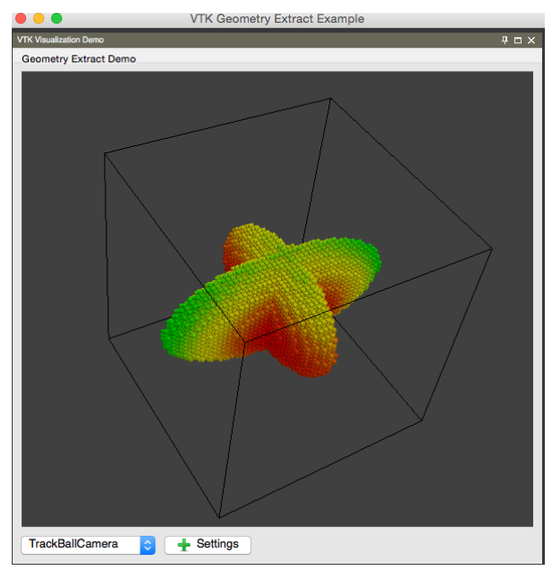
VTK Examples



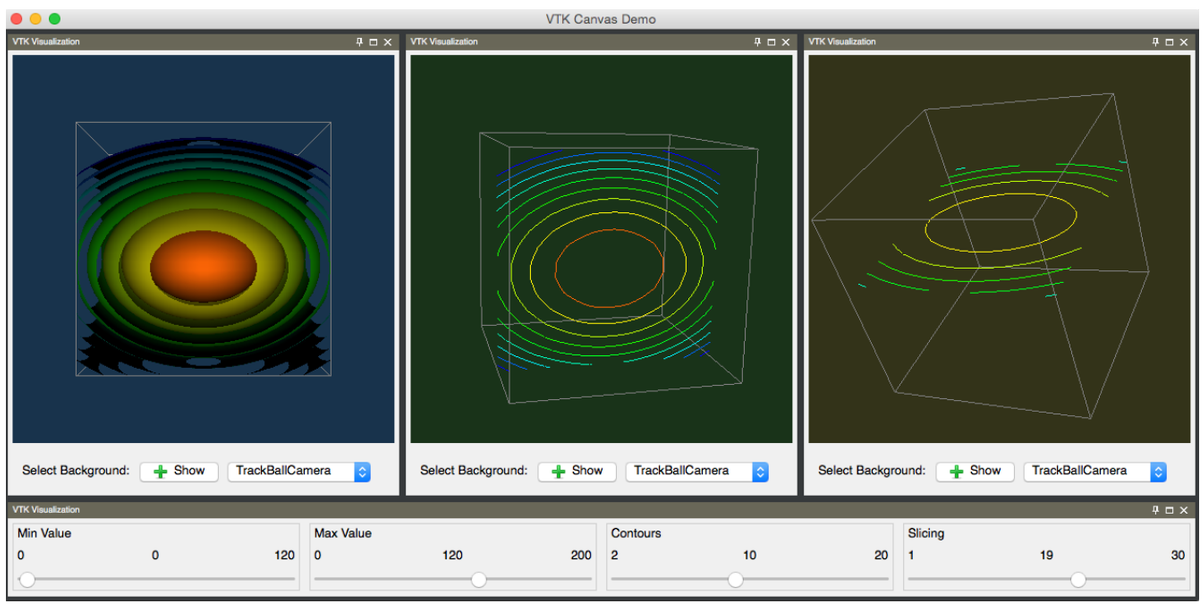
vtk_surface_view.enaml



vtk_sample_grid_view.enaml



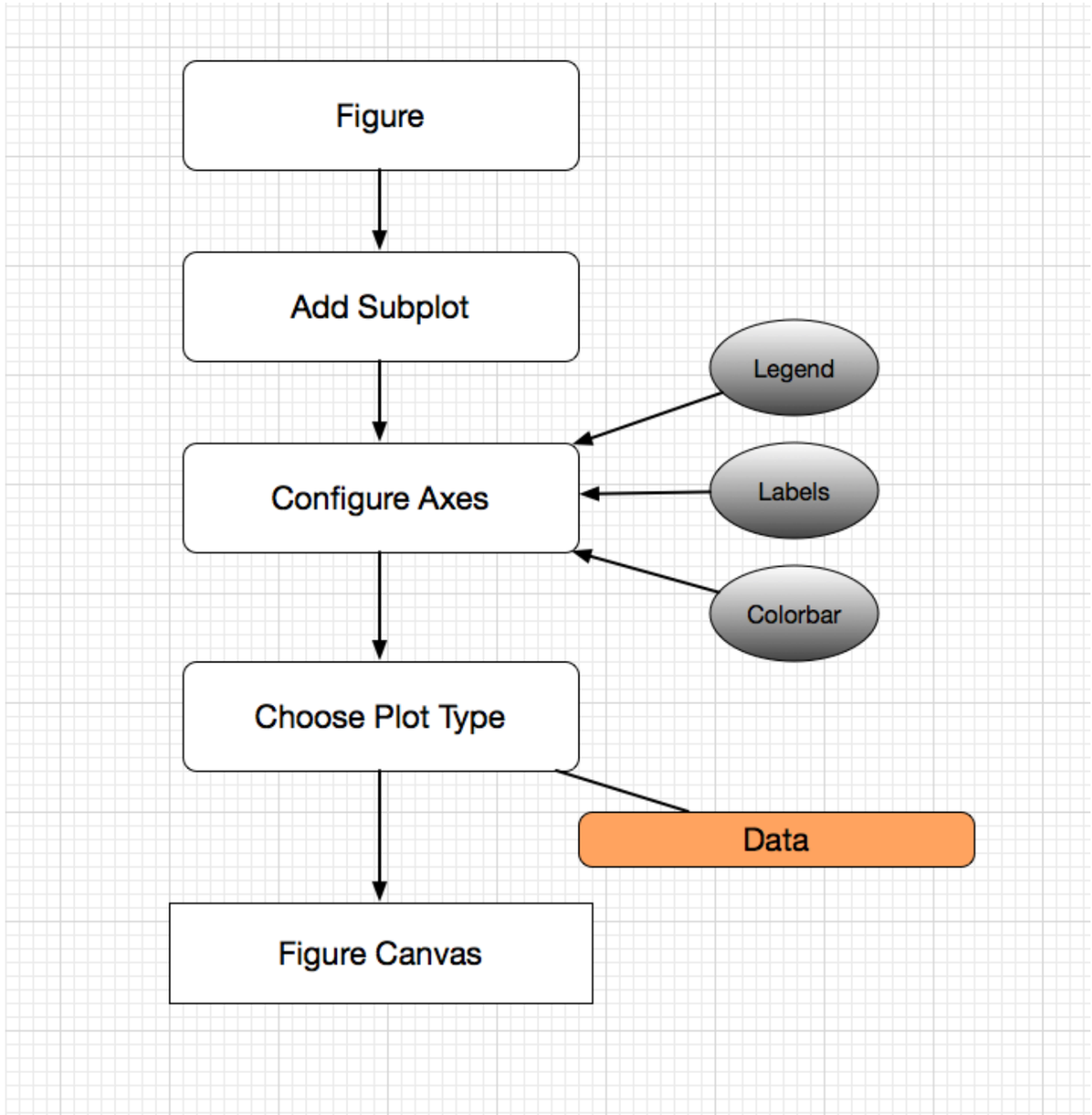
vtk_extract_view.enaml



vtk_multi_windows_view.enaml

Matplotlib - Use Figure to expose drawing parameters to Enaml application

- With Enaml we are creating Figure to specify plot type and number of plots
- Figure is feed into FigureCanvas - wrapped in Enaml - QT backend
- Add Sub Plot to the Figure
- Modify Axes properties
- Create data set
- Choose plot type and plot data



Matplotlib - extending widgets creating custom Components

- To extend MPL Canvas functionality and to reuse code create custom MPL Canvas Container

- Use Enaml features to extend and wrap MPL canvas

- Expose new attributes

- Extend existing properties

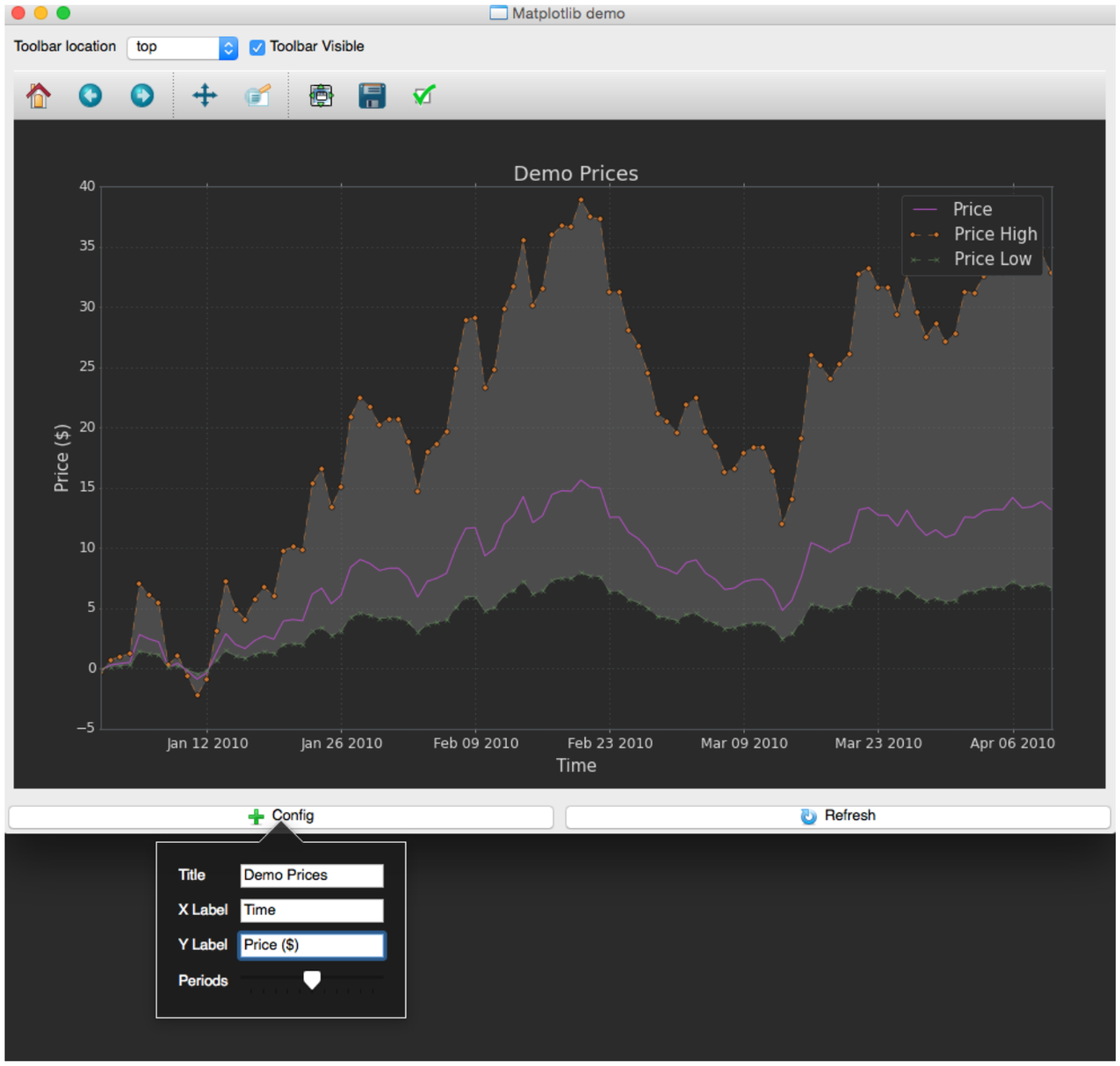
```
enamldef MPLCanvasContainer( Container ):

    attr mpl_figure
    attr mpl_event_actions

    padding = 5
    constraints = [
        vbox(
            hbox(l, obox, check, spacer),
            canvas,
        ),
        align( 'v_center', l, obox, check )
    ]

Label: l:
    text = 'Toolbar location'

ObjectCombo: obox:
    items = [ 'top', 'bottom' ]
    selected = 'bottom'
    selected ::
        canvas.toolbar_location = selected

CheckBox: check:
    text = 'Toolbar Visible'
    checked := canvas.toolbar_visible

MPLCanvas: canvas:
    figure << mpl_figure
    toolbar_location << obox.selected
    event_actions << mpl_event_actions
```

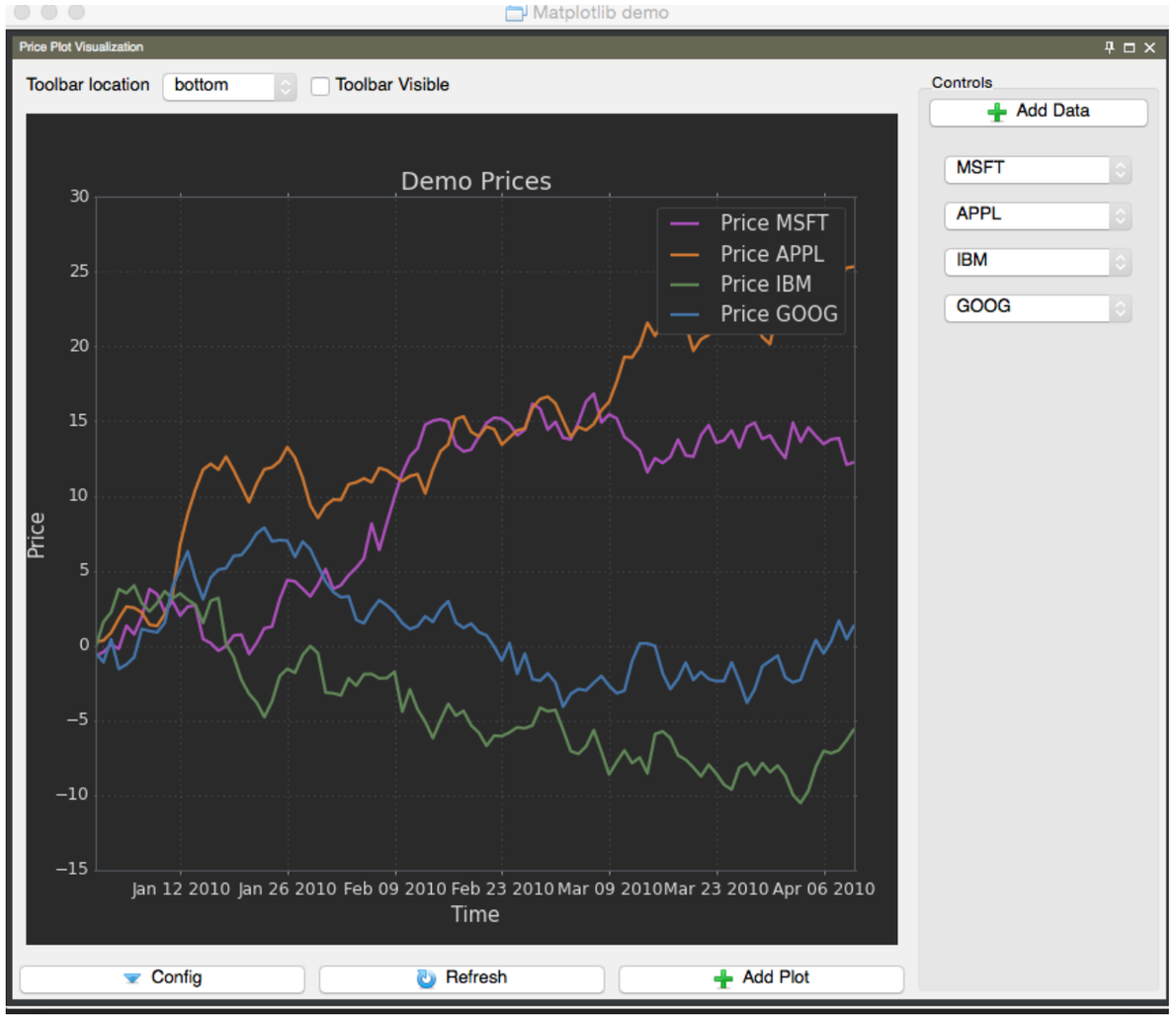
```
enamldef Main(Window):

    attr custom_title = ''
    title = custom_title
    attr mplot_style = ''

    MPLCanvasContainer: mpl:
        mpl_figure << get_figure..
        mpl_event_actions << [ ( 'key_press_event', on_key_press ), ]

    initialized ::
        style.use( mplot_style )
```

Use MPL Canvas container and Atom based model to control plot attributes

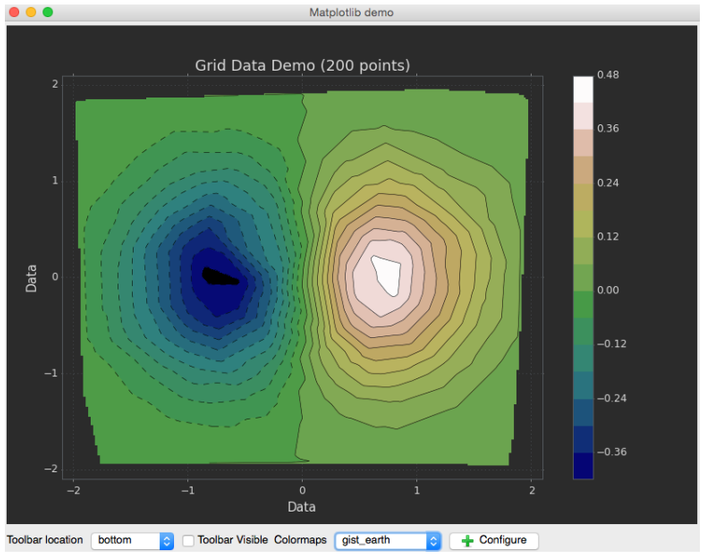
```
def on_key_press( *args, **kwargs ):
    """ on key press event """
    print '>>> press ', args, kwargs


def plot_lines( plt, params ):
    x = np.linspace(0, 10)
    shape = 50

    with style.context( 'fivethirtyeight' ):
        for param in params:
            plt.plot(x, np.sin(x) + param * x + np.rand

def get_figure( shape=50, params=( 1, 0.5, 2 ), title='
    """ create Figure """

    with style.context( 'fivethirtyeight' ):
        fig = Figure()
        plt = fig.add_subplot(1, 1, 1)

        plot_lines( plt, params )
        fig.suptitle( title )
        return fig, plt


enamldef Main(Window):

    attr custom_title = ''
    title = custom_title
    attr mplot_style = ''

    attr scale_factor = [ 1, 0.5, 2 ]
    attr figure_params = get_figure( params=scale_facto
    attr controller = ModelController( canvas=mpl.canva

    Container:
        padding = 5
        constraints = [ vbox( mpl, b ) ]

        MPLCanvasContainer: mpl:
            mpl_figure << figure_params[0]
            mpl_event_actions << [ ( 'key_press_event',

        PushButton: b:
            text = 'Config'
            icon = load_icon('plus.png')
            clicked ::
                ConfigPopup( self ).show()

    initialized ::
        style.use( mplot_style )
```

Matplotlib and Enaml Interaction - Examples

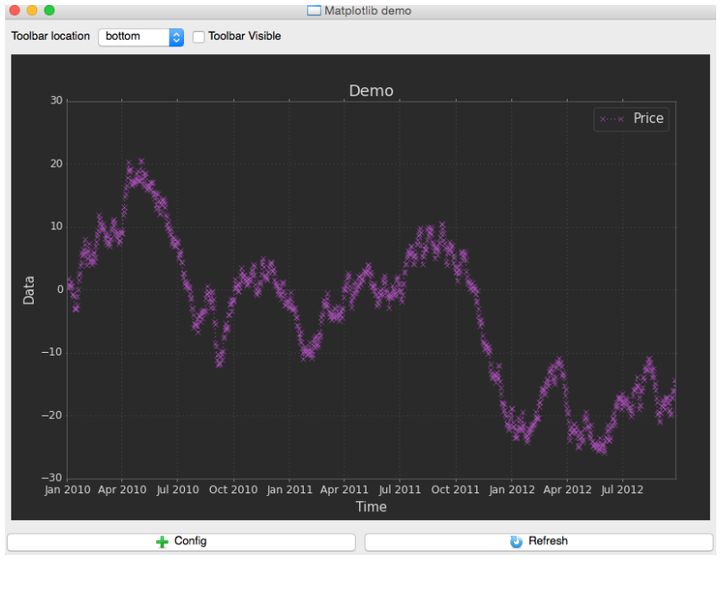




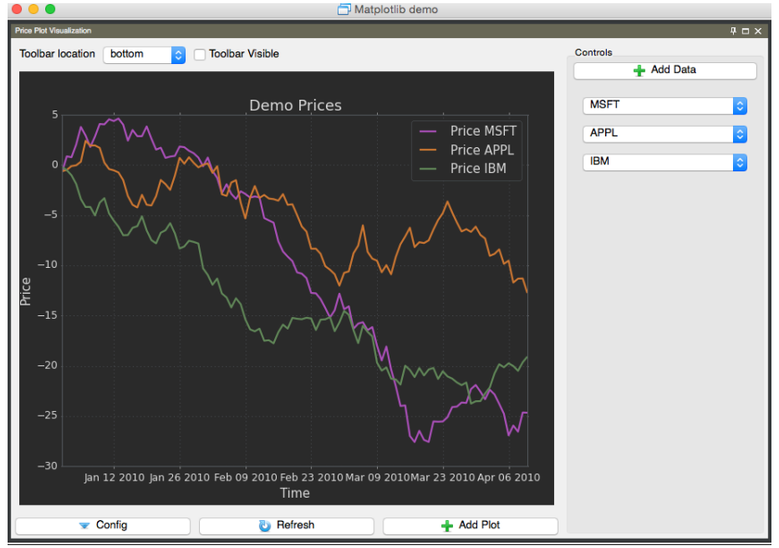
Matplotlib Examples



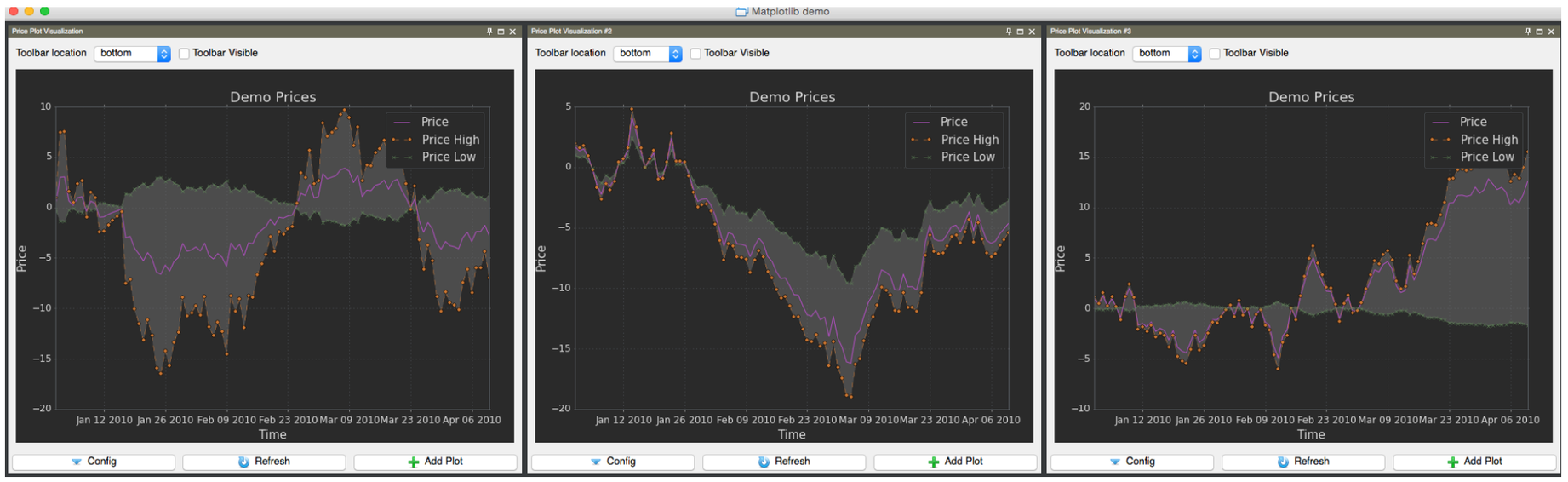
griddata_demo_model_ui



matplot_demo_dynamic



matplot_demo_controls



matplot_demo_docks

Conclusion

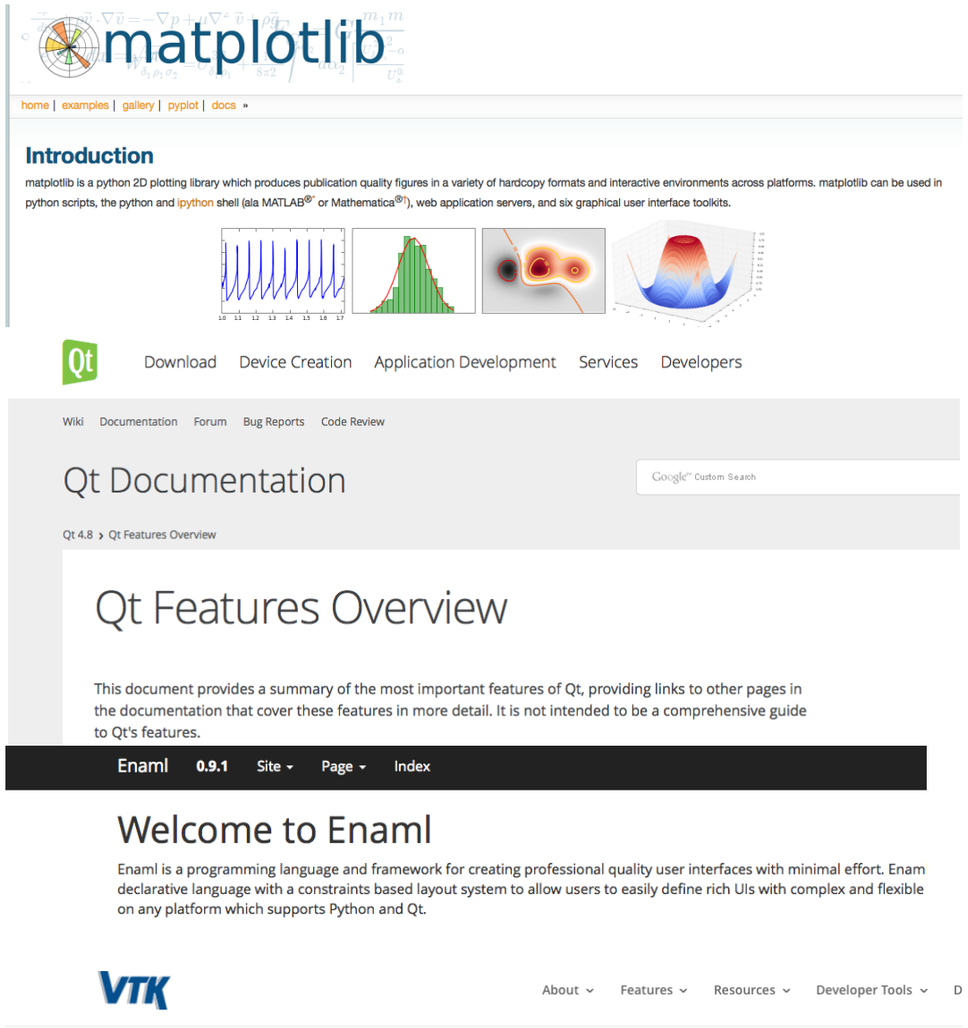
- Hopefully I demonstrated how easy can be UI development with Enaml
- Building high quality interactive applications build with Enaml
- Demonstrated VTK integration with Enaml by modifying VTK pipeline
- Demonstrated Matplotlib integration with Enaml by modifying Figure and Axes objects

Future

- Implement more of core QT widgets in Enaml
- Add real support for various IDE's like PyCharm
- Extend Enaml for other backends (web)
- VTK - extend Enaml to provide declarative way of creating visualizations

```
enamldef DemoScene( VTKRenderer ):

    background = '#000000'

    VTKActor:

        properties = [ color = '#cdcdcd', ]

    VTKPolyDataMapper:

        scalar_range = ( 0.0, 1.2 )

    VTKContourFilter:

        VTKSampleFunction:

            dimension = ( 50, 50, 50 )

            VTKQuadric:

                    coefficients = ( .5, 1, .2, 0, .1, 0, 0, .2, 0, 0 )
```

Resources



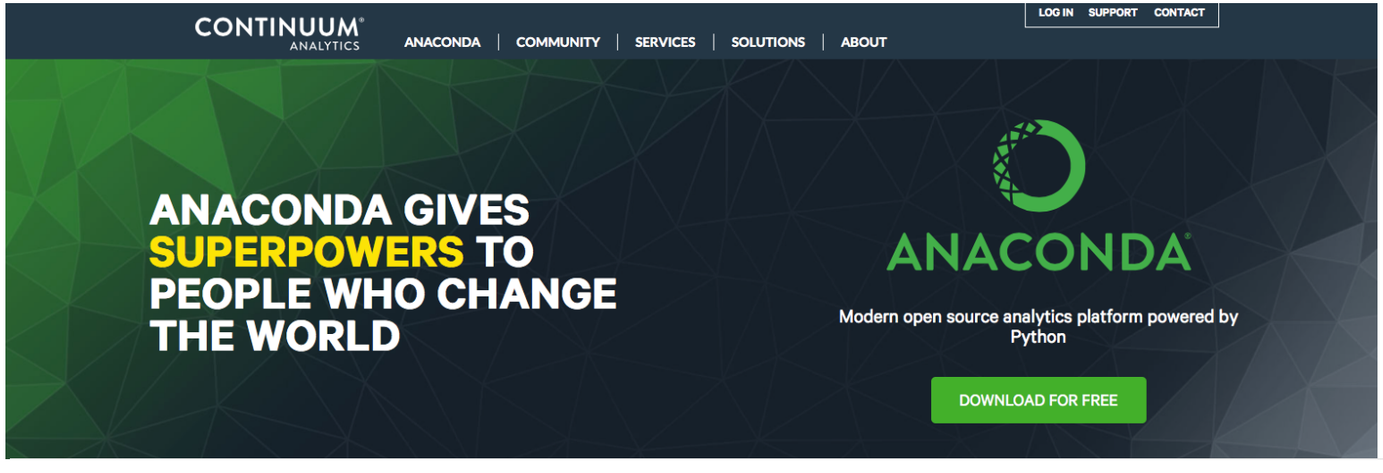
www.matplotlib.com

http://doc.qt.io/qt-4.8

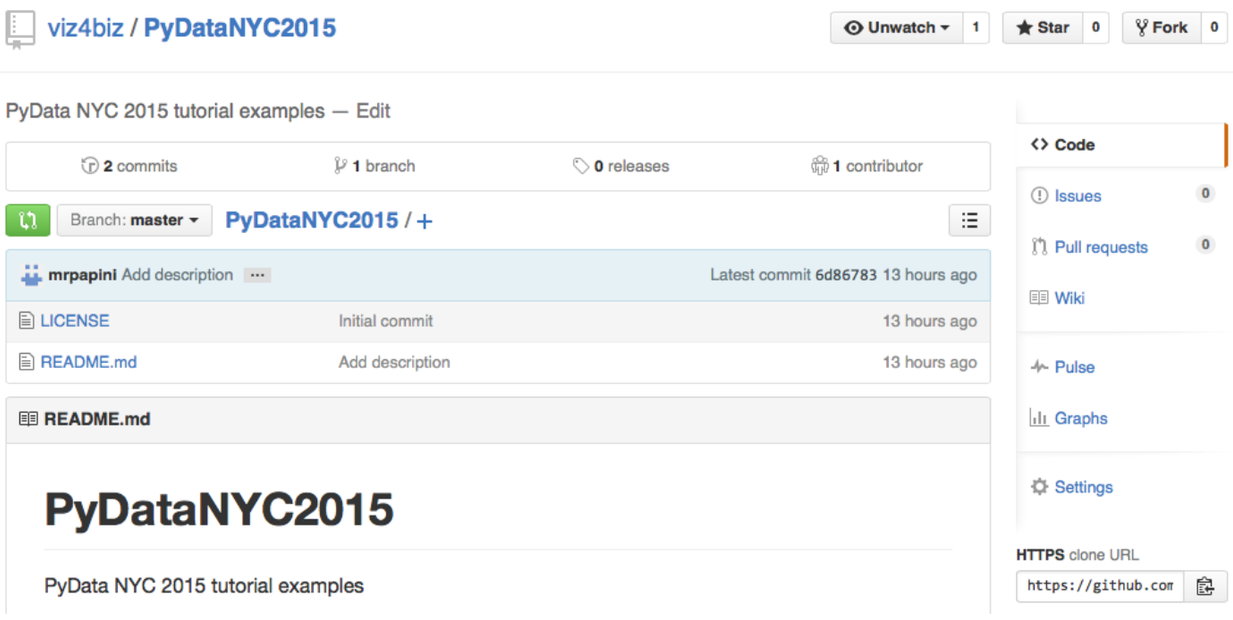
http://nucleic.github.io/enaml/docs/

http://www.vtk.org

Resources



https://www.continuum.io



https://github.com/viz4biz/PyDataNYC2015

Disclaimer