Interactive Visualization Widgets Using Chaco and Enable

Corran Webster
Enthought

The Software Stack

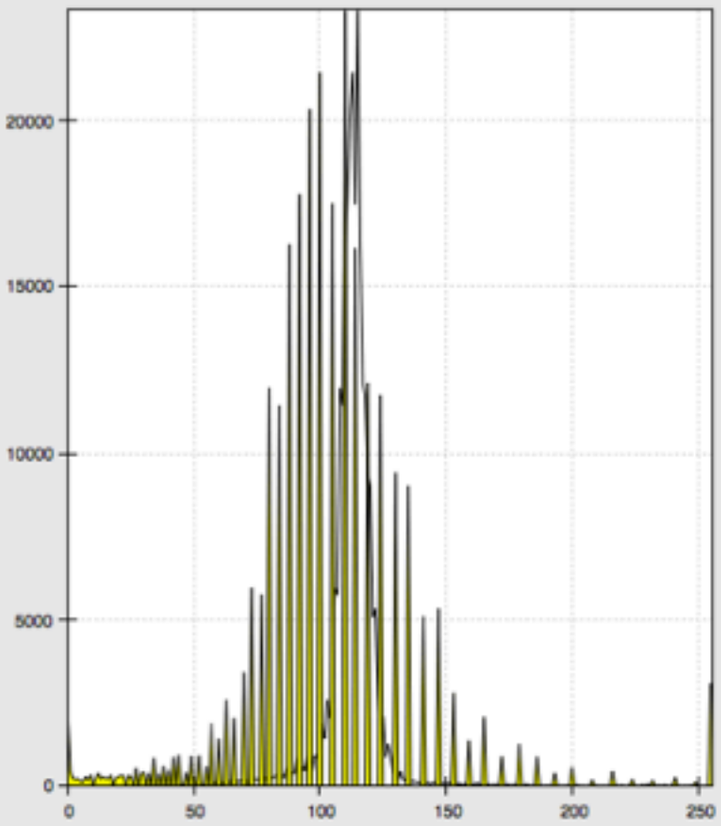
Chaco
(plotting)

Enable
(interactive drawing)

Traits
(notification)

Kiva
(vector drawing)

Edit properties



Low:	0.0	1.0	0.1048
High:	0.0	1.0	1.0
Slope:	0.1	10.0	1.99783
Intercept:	-1.0	1.0	-0.0666
Gamma:	0.1	10.0	4.555

Above all else show the data.

Maximize the data-ink ratio.

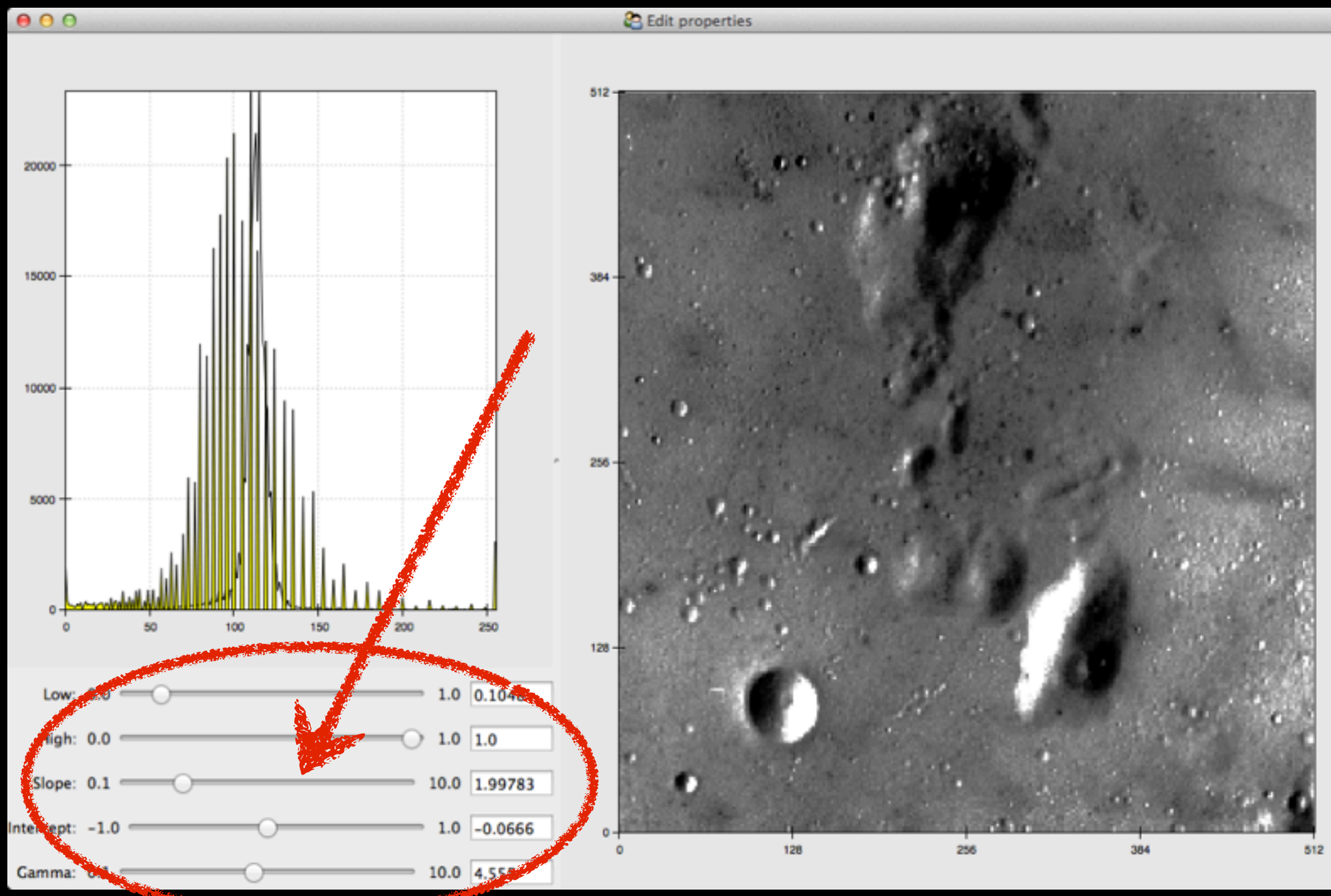
Erase non-data-ink.

Erase redundant data-ink.

Revise and edit.

Edward R. Tufte,

The Visual Display of Quantitative Information


Edit properties
20000
15000
10000
5000
0
0
50
100
150
200
250
Low: 0.0
1.0
0.1048
High: 0.0
1.0
1.0
Slope: 0.1
10.0
1.99783
Intercept: -1.0
1.0
-0.0666
Gamma:
10.0
512
384
256
128
0
0
128
256
384
512

Above all else show the data.

Maximize the data-~~ink~~ pixel ratio.

Erase non-data-~~ink~~ pixels.

Erase redundant data-~~ink~~ pixels.

Revise and edit.

Edward R. Tufte,

The Visual Display of Quantitative Information

Clean Up The Plots

- Remove axes and grids:

```
plot = Plot(self.plot_data)
plot.x_axis = None
plot.y_axis = None
plot.x_grid = None
plot.y_grid = None
```

- Remove padding:

```
plot.padding = 0
```

Write Chaco Tools

- Very tempting to write one tool to rule them all (see BetterSelectingZoom tool)
- Interactions become hard to track, debug
- Getting tools to work together becomes complex, fragile
- Better to use multiple simple tools

AttributeDragTool

- Use the new AttributeDragTool in Enable

```
intercept_tool = AttributeDragTool(
    component=plot,
    model=self.unit_map,
    x_attr='intercept')
plot.tools.append(intercept_tool)
```

- AttributeDragTool is mapper aware so that dragging can be in data space
- Simply modifies a Trait by a delta

Better Interactions

- Subclass AttributeDragTool to interact with plots:

```
class PlotDragTool(AttributeDragTool):
    #: the plot we are targeting - this can be
    #: anything which implements a hittest() method
    plot = Any

    def is_draggable(self, x, y):
        return self.plot.hittest((x, y))
```

Feedback with Overlays

- SimpleInspectorOverlay gives a basic text display of a value

```
intercept_overlay = SimpleInspectorOverlay(
     component=plot, align='ul',
     inspector=intercept_tool,
     field_formatters=[
         [basic_formatter('Intercept', 2)]
     ]
 )
 plot.overlays.append(intercept_overlay)
```

Feedback with Custom Overlays

- Frequently you can indicate values by drawing on the plot
- For example, high and low values can be shown by simple vertical lines
- We can write an overlay which looks at a trait and draws a line at the specified point.

Feedback with Custom Overlays

```
def overlay(self, gc):
    sx = self.x_mapper.map_screen(self.value)
    with gc:
        gc.clip_to_rect(self.component.x,
            self.component.y,
            self.component.width,
            self.component.height)
        gc.set_stroke_color(self.line_color_)
        gc.set_line_width(self.line_width)
        gc.set_line_dash(self.line_style_)
        gc.move_to(sx, self.component.y)
        gc.line_to(sx, self.component.y2)
        gc.stroke_path()
```

Feedback with Custom Overlays

- Implimenting the hittest() function, then allows the PlotDragTool to work:

```
def hittest(self, screen_pt, threshold=7.0):
    x, y = screen_pt
    x_value, y_value = self.get_value()

    sx = self.x_mapper.map_screen(x_value)
    return abs(x-sx) <= threshold
```

Rules to Follow

- Maximize the data-pixel ratio
- Simple tools and overlays that do one thing and one thing only
- Don't be afraid to get your hands dirty writing Enable/Chaco drawing code
- Above all else show data

Resources

Most of these tools and overlays will be cleaned up and added to Chaco or Enable

Example code is on github:

github.com/corranwebster/scipy-2012

cwebster@enthought.com