plot_controls

March 30, 2024

1 PlotControls adjust how a *design* is transformed into a 'plot' *result*

designs are transformed into a 'plot' according to some default settings which can be overwritten with a PlotControls object with the following attributes (all demonstrated in this notebook):

- `color_type` - options: 'random_blue', 'z_gradient', 'print_sequence', 'print_sequence_fluctuating', 'manual'
- `zoom` - recommended range 0.1 - 10
- `hide_annotations` - True/False
- `hide_travel` - True/False
- `hide_axes` - True/False
- `neat_for_publishing` - True/False (square format for consistent png creation)
- `raw_data` - True/False (output data instead of creating a plot)
- `style` - options: 'tube'/'line' - preview 3D real-printed-volume lines or simple lines with non-representative widths
  - if `style == 'tube'`:
    - `tube_type` - options: 'flow'/'cylinders' - adjust how the plot transitions from line to line
    - `tube_sides` - sides of the prisms created for the 3D real-printed-volume lines
    - `initialization_data` - define initial width/height of 3D lines with dictionary: {'extrusion_width': value, 'extrusion_height': value}
  - if `style == 'line'`:
    - `line_width` - recommended range 0.1 - 10

custom plots can be created with the raw data as demonstrated below

<this document is a jupyter notebook - if they're new to you, check out how they work: link, link, link>

run all cells in this notebook in order (keep pressing shift+enter)

[ ]:
```
import fullcontrol as fc

# this demo design is used for most of the plots in this notebook
centre_point = fc.Point(x=50, y=50, z=0)
steps = fc.helixZ(centre_point, 20, 15, 0, 30, 0.15, 30*64)
steps.append(fc.Extruder(on=False))
steps.append(fc.PlotAnnotation(label='extruder off'))
```

```
steps.append(fc.Point(x=50, y=50, z=0))
steps.append(fc.Extruder(on=True))
steps.append(fc.PlotAnnotation(label='extruder on'))
steps.append(fc.Point(z=5))
steps.append(fc.PlotAnnotation(label='finish'))
steps.append(fc.PlotAnnotation(point=steps[0], label='start'))
```

default plot style lines are shown as 3D lines, where width and height are defined by the extrusion width and height set in initialization data or through ExtrusionGeometry objects in the ***design*** - if unset, default values are width 0.4 and height 0.2

[ ]:
```
fc.transform(steps, 'plot')
```

change color and hide axes / annotations / travel

[ ]:
```
plot_controls = fc.PlotControls(line_width=4, color_type='print_sequence',␣
  ↪hide_axes=True, hide_annotations=True, hide_travel=True)
fc.transform(steps, 'plot', plot_controls)
```

'neat_for_publishing' and 'zoom' create a square plot for consistent generation of images for publication with neat_for_publishing=True

hover the mouse over the plot and click the camera button ("Download plot and png")

'zoom' is used to set the initaial zoom level of the plot

[ ]:
```
fc.transform(steps, 'plot', fc.PlotControls(neat_for_publishing=True, zoom=0.6))
```

change line size extrusion width and height are controlled in the ***design*** with ExtrusionGeometry objects, as discussed in the State Objects notebook

these objects are automatically evaluated - they dictate the widths and heights of lines in the plot

the plot can be initated with specific widths and heights by including an 'initialization_data' dictionary in PlotControls (first example below)

the specified width and height remain throughout the plot unless the design includes ExtrusionGeometry objects (second example below)

if no intitialization data is provided, default values are used: 0.4 wide x 0.2 high

[ ]:
```
plot_controls = fc.PlotControls(initialization_data={'extrusion_width': 0.1,␣
  ↪'extrusion_height': 0.1})
fc.transform(steps, 'plot', plot_controls)
```

'tube_type' set to 'flow' to get smooth transitions between linear segments of the path - this leads to much neater visuals for curves and allows gradual transitions when extrusion width is changed

set to 'cylinders' to get a more strict preview of widths defined in the ***design*** - each tube has a contant width according to the designed width

```
[ ]: varying_width_steps = []
varying_width_steps.append(fc.Point(x=0, y=5, z=0.1)) # start point (width␣
 ↪defaults to 0.4)
varying_width_steps.append(fc.ExtrusionGeometry(width=0.6, height=0.4))
varying_width_steps.append(fc.Point(y=0)) # print to this point with width 0.6
varying_width_steps.append(fc.ExtrusionGeometry(width=1))
varying_width_steps.append(fc.Point(x=5)) # print to this point with width 1
varying_width_steps.append(fc.ExtrusionGeometry(width=1.5))
varying_width_steps.append(fc.Point(y=5)) # print to this point with width 1.5
fc.transform(varying_width_steps + [fc.PlotAnnotation(label='tube_type="flow"␣
 ↪(view from above to see clearly)')], 'plot', fc.
 ↪PlotControls(color_type='print_sequence', tube_type="flow")) # default␣
 ↪tube_type="flow"
fc.transform(varying_width_steps + [fc.
 ↪PlotAnnotation(label='tube_type="cylinders" (view from above to see␣
 ↪clearly)')], 'plot', fc.PlotControls(color_type='print_sequence',␣
 ↪tube_type="cylinders"))
```

manual colors set color_type='manual' and assign [r, g, b] colors to points for manual colors

any points without the attribute 'color' defined will inherit the color of the previous point

colors automatically transition over the length of a line between points with different colors

```
[ ]: colored_steps = []
colored_steps.append(fc.Point(x=0, y=0, z=0.2, color=[1, 0, 0]))
colored_steps.append(fc.Point(x=40, y=4, color=[1, 0.8, 0]))
colored_steps.append(fc.Point(x=0, y=8))
fc.transform(colored_steps, 'plot', fc.PlotControls(color_type='manual'))
```

'tube_sides' extuded lines are plotted as 6-sided hexagonal prisms by default, but the number of sides can be change

```
[ ]: steps_line = [fc.Point(x=0, y=0, z=0.1), fc.Point(y=1), fc.Point(y=2)]
fc.transform(steps_line + [fc.PlotAnnotation(label='8-sided tube',␣
 ↪point=steps_line[0])], 'plot', fc.PlotControls(color_type='print_sequence',␣
 ↪style="tube", tube_sides=8))
```

plot_style (line) as opposed to plotting the path as 3D lines with real volumes representing the width and height of extruded lines, it is possible to create a simple line plot

zooming in and out of a line plot does not change the line size and it does not represent the width or height of printed lines

the width of the line can be controlled with the 'line_width' attribute

however, this type of plot has the advantage that it is less computationally demanding, which may be important for larger models

also, this plot is useful for design since the mouse cursor can be used to identify coordinates of points directly on the print path (nozzle position) as opposed to identiying points on the surface of the 3D line preview (tublular structures)

[ ]:
```
fc.transform(steps + [fc.PlotAnnotation(point=fc.Point(x=50, y=50, z=15),␣
 ↪label='zoom in to see line width remain constant')], 'plot', fc.
 ↪PlotControls(style='line', line_width=2))
```

output and inspect raw data

[ ]:
```
plot_controls = fc.PlotControls(raw_data=True)
plot_data = fc.transform(steps, 'plot', plot_controls)
print('first five values of the first path:')
print(f'    x values: {plot_data.paths[0].xvals[0:4]}')
print(f'    y values: {plot_data.paths[0].yvals[0:4]}')
print(f'    z values: {plot_data.paths[0].zvals[0:4]}')
print(f'    extrusion width values: {plot_data.paths[0].widths[0:4]}')
print(f'    extrusion height values: {plot_data.paths[0].heights[0:4]}')
print(f'    color values [r, g, b]: {plot_data.paths[0].colors[0:4]}')
print(f'    extruder state: {plot_data.paths[0].extruder.on}')
print(f'second path (travel line of two points):\n    {plot_data.paths[1]}')
print(f'final path (vertical line of two points):\n    {plot_data.paths[2]}')
print(f'plot_data.annotations:\n    {plot_data.annotations}')
print(f'plot_data.bounding_box:\n    {plot_data.bounding_box}')
```

create custom plots this is the same path as in previous plots but this plot doesn't scale xyz axes equally

[ ]:
```
def custom_plot(data):
    import plotly.graph_objects as go
    fig = go.Figure(layout=go.Layout(template='plotly_dark'))
    for i in range(len(data.paths)):
        line_color = 'rgb(255,160,0)' if data.paths[i].extruder.on == True else␣
 ↪'rgb(200,0,0)'
        fig.add_trace(go.Scatter3d(mode='lines', x=data.paths[i].xvals, y=data.
 ↪paths[i].yvals,z=data.paths[i].zvals, line=dict(color=line_color)))
    fig.show()

plot_controls = fc.PlotControls(raw_data=True)
plot_data = fc.transform(steps, 'plot', plot_controls)
custom_plot(plot_data)
```